PROYECTO DE SISTEMAS INFORMÁTICOS
2008/2009

FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

SISTEMA DE ACCESO BLUETOOTH

AUTORES:
Jesús María Cano García
Teresa García de la Torre
Pedro Sánchez Aparicio

PROFESOR DIRECTOR:
Luis Piñuel Moreno
Dep. Arquitectura de Computadores y Automática

Agradecimientos

Agradecemos a nuestras familias el apoyo que nos han prestado durante todos estos años de carrera. Soportarnos y ayudarnos en los momentos de mayor desesperación no es nada fácil y ellos lo han hecho siempre y de manera incondicional.

También queremos agradecer la ayuda que nos han prestado de forma totalmente desinteresada los técnicos tanto de la facultad de Físicas como de la de Informática.

Y, cómo no, acordarnos de nuestros amigos y compañeros, sin los cuales el camino hasta aquí no habría sido el mismo.

Derechos

Listado de palabras clave

- AES
- Arduino
- Bluetooth
- Ethernet
- J2ME
- LDAP
- Móviles
- Seguridad
- Sockets
- UDP

Índice de contenidos

Introducción

Identificación del proyecto

Nombre

El título original que describe de forma fiel el proyecto es “Sistema de Control de Acceso Físico mediante Bluetooth”, pero debido a su longitud se decidió simplificar a “Sistema de Acceso Bluetooth”.

Acrónimo

Por lo explicado anteriormente el acrónimo es **SAB**, “Sistema de Acceso Bluetooth”.

Logotipos

SAB 

Resumen

Los sistemas de Control de Acceso Físicos son un pilar imprescindible dentro de los sistemas de seguridad puesto que se encargan de permitir o cancelar el paso a un espacio restringido. En la actualidad, cualquier tipo de institución, cuando se plantea la instalación de un sistema de control de acceso, no busca sólo impedir los accesos no deseados a sus instalaciones, lo que desea es que se le garantice que el acceso a sus recintos además de ser seguro se realice de forma cómoda y organizada. Los avances tecnológicos han permitido que se puedan sustituir las cerraduras y llaves comunes por sistemas electrónicos que además de conferir mayor seguridad ofrecen un amplio campo de posibilidades. Algunos de estos sistemas son equipos de identificación de personal basados en tarjetas HID, tarjetas con código de barras, equipos biométricos, tarjetas de banda magnética, teclados, botoneras, etc.

El objetivo de este proyecto es el diseño e implementación de un sistema de control de acceso físico alternativo a los mencionados previamente. Un sistema fácil de usar, cómodo y económico, que proporcione una seguridad configurable, una gestión de ésta centralizada y que permita a los usuarios utilizar sus propios teléfonos móviles como llaves de acceso a las áreas restringidas. Podemos diferenciar tres bloques principales: un sistema empotrado para el que hemos utilizado una plataforma de hardware libre y de fuente abierta, un servidor con un directorio LDAP, desde el que se gestiona la seguridad de cada área, y el dispositivo de identificación personal, un teléfono móvil que establece la conexión pertinente mediante Bluetooth.

Abstract

The Physical Access Control systems are a fundamental pillar of the security systems because they are responsible for allowing or cancel the acces to a restricted area. Nowadays, when any company considers to install an access control system, doesn't do it only with the aim of avoid not desired access, but want a secure, organized and convenient access to the restricted zones. Advanced Technology has allowed to replace common locks and keys with electronic systems that in addition to greater security offer a wide range of possibilities. Some of these systems are personal identification equipments based on HID cards, bar code cards, biometric equipments, magnetic stripe cards, keyboards, buttons, etc.

The aim of this project is to design and implement an alternative Physical Access Control system. One easy to use, convenient and economical, that provides configurable security and centralized management, and wich allows users to use their own mobile phones as keys for the restricted areas. We can distinguish three main blocks: an embedded system for which we have used open-hardware and open-source platform, a server with an LDAP directory, which manages the security of each area, and the personal identification device, a phone that establish the Bluetooth connection.

Objetivos

Este proyecto pretende, como se ha especificado anteriormente, la obtención de un sistema de control de acceso físico para puertas. Nos hemos basado para ello en la utilización de la tecnología Bluetooth dada su integración actual en dispositivos de la vida cotidiana, como pueden ser los teléfonos móviles.

Está destinado a entidades que buscan una seguridad organizativa, centralizada y sencilla, huyendo de los altos precios que suponen opciones ya mentadas como las tarjetas HID o equipos biométricos.

Planificación

Se comenzará con un proceso de investigación para la elección de la tecnología, el entorno y la plataforma más adecuados en cada caso, intentando en la medida de lo posible utilizar hardware y software libre.

Una vez seleccionados todos los elementos a usar se realizarán los prototipos necesarios y las pruebas oportunas que verifiquen la viabilidad de los objetivos establecidos.

A continuación se procederá al diseño e implementación del producto final, acompañados éstos de las pruebas correspondientes.

Finalmente se realizarán las mejoras que sean factibles.

Capítulo 1. Bluetooth

1.1. Origen e historia de Bluetooth

Harald Bluetooth fue un rey vikingo que reinó en Dinamarca y Noruega en el siglo X (años 960-986). Harald fue un rey muy activo e inusual, tanto físicamente como en su actitud: aunque la vida de los vikingos consistía en pelear y saquear, Harald consiguió tanto implantar el cristianismo en Escandinavia como unificar Noruega y Dinamarca. Mil años más tarde, en 1994, nacía en Suecia una nueva tecnología de comunicaciones inalámbricas, que fue llamada Bluetooth en honor al gran rey Harald.

Bluetooth fue introducido por Ericsson, IBM, Intel, Nokia y Toshiba a comienzos de 1998. Estas compañías posteriormente formaron un grupo de interés especial conocido como el Bluetooth SIG. El lanzamiento de las especificaciones de Bluetooth 1.0 se realizó el 26 de julio de 1999, pero sólo recientemente la tecnología se ha vuelto lo suficientemente económica como para ser usada de manera generalizada.

1.2. Definición

Bluetooth es una tecnología de comunicaciones inalámbrica de corto alcance que opera en el ancho de banda libre de ISM (banda internacional medico-científica), a 2.4 GHz. Su máxima velocidad de transmisión de datos es de 1 Mbps. El propósito de esta nueva tecnología es la comunicación entre dispositivos a pequeñas distancias (hasta 100 metros).

1.3. Bluetooth e infrarrojos

La comparación entre estas dos tecnologías es muy simple: mientras que en la comunicación por infrarrojos es necesario que los dispositivos tengan visibilidad óptica entre

ellos, el sistema Bluetooth, al ser una tecnología de comunicación por radio-frecuencia, no requiere esta visibilidad.

1.4. Bluetooth y 802.11b (Wifi)

Mientras que la tecnología 802.11b está diseñada para la comunicación entre dispositivos de gran tamaño y potencia (portátiles, PC...) y para una transmisión de datos a gran velocidad (11 Mbps) y a grandes distancias, Bluetooth está pensado para todo lo contrario: para la comunicación entre dispositivos pequeños y limitados en potencia; de ahí su lenta velocidad de transmisión (1 Mbps) y su corto alcance.

1.5. Saltos de frecuencia

Dado que la red ISM es una banda libre, el sistema Bluetooth tiene que estar preparado para evitar las interferencias que se puedan producir. Para ello, usa un sistema de salto en frecuencia.

Mediante esta técnica, la banda de frecuencia se divide en varios canales de salto, donde los transceptores, durante la comunicación, van saltando de un canal a otro de forma pseudo-aleatoria. Estos saltos se producen a una gran velocidad (1600 saltos/segundo). Los paquetes están protegidos con un sistema ARQ (repetición automática de consulta), de modo que si algún paquete se pierde se vuelve a retransmitir.

1.6. Canales

El sistema Bluetooth usa FH/TDD (Saltos en frecuencia/División de tiempo dúplex) de esta forma el canal queda dividido en slots o intervalos de 625 μs y cada salto de frecuencia es ocupado por uno de estos intervalos.

Dos o más dispositivos Bluetooth pueden compartir el mismo canal dentro de lo que se llama una piconet (pequeña red establecida entre terminales Bluetooth para comunicarse entre ellos), donde una de las unidades funciona como unidad maestra y se encarga de controlar el tráfico de datos que se genera entre las demás unidades. Las unidades restantes actúan como esclavas, enviando y recibiendo señales hacia la unidad maestra.

El salto de frecuencia del canal queda determinado por la secuencia de la señal, es decir, por el orden en que llegan los saltos y por la fase de esta secuencia. En las comunicaciones Bluetooth, esta secuencia queda determinada por la identidad de la unidad maestra (dentro de la piconet) y por su frecuencia de reloj.

1.7. Piconet y Scatternet

Como se comentaba en el apartado anterior, una **piconet** es una pequeña red Bluetooth compuesta por un dispositivo que hace las veces de maestro y uno o más que las hacen de esclavos. El dispositivo que inicializa la conexión Bluetooth automáticamente se convierte en un maestro. Una piconet puede estar compuesta por un maestro y hasta siete esclavos.

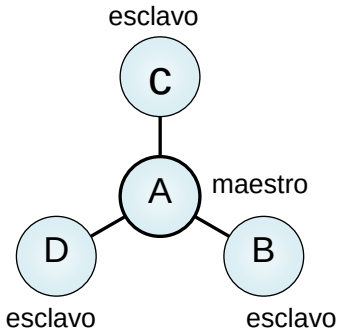


Figura 1. 1. *Piconet.*

El dispositivo que funciona como maestro es literalmente el maestro de la piconet, los esclavos sólo transmiten información cuando el dispositivo maestro se lo permite. Los dispositivos esclavos no pueden comunicarse entre ellos directamente, para ello deben hacerlo a través del maestro. Los esclavos sincronizan sus saltos de frecuencia con el maestro, usando su frecuencia de reloj y su dirección Bluetooth.

Las piconets tienen forma de red estrella, donde el dispositivo maestro es el centro del nodo. Dos piconets pueden coexistir en un mismo radio de acción. Los saltos de frecuencia no se sincronizan entre las piconets, por lo que diferentes piconets colisionarán de forma aleatoria en la misma frecuencia.

Cuando se conectan dos piconets, el resultado será una **scatternet**. En este sistema, un nodo intermedio conecta las piconets. Este nodo intermedio debe repartirse el tiempo, es decir, sólo puede hacer saltos de frecuencia en una piconet a la vez. Esto reduce la cantidad de intervalos posibles para la transmisión de datos entre el nodo intermedio y el nodo maestro, por lo que la velocidad de transmisión quedará mermada a la mitad.

1.8. Arquitectura Bluetooth

1.8.1. Pila de protocolos Bluetooth

Como hemos visto en apartados anteriores, la especificación Bluetooth permite a los dispositivos Bluetooth (de diferentes fabricantes) comunicarse entre sí. No basta con especificar sólo el sistema de comunicaciones por radio. Por ello, la especificación Bluetooth no está basada sólo en el sistema radio, sino que además hay una compleja pila de protocolos detrás que aseguran que los dispositivos Bluetooth puedan comunicarse entre ellos, explorar sus servicios y hacer uso de ellos.

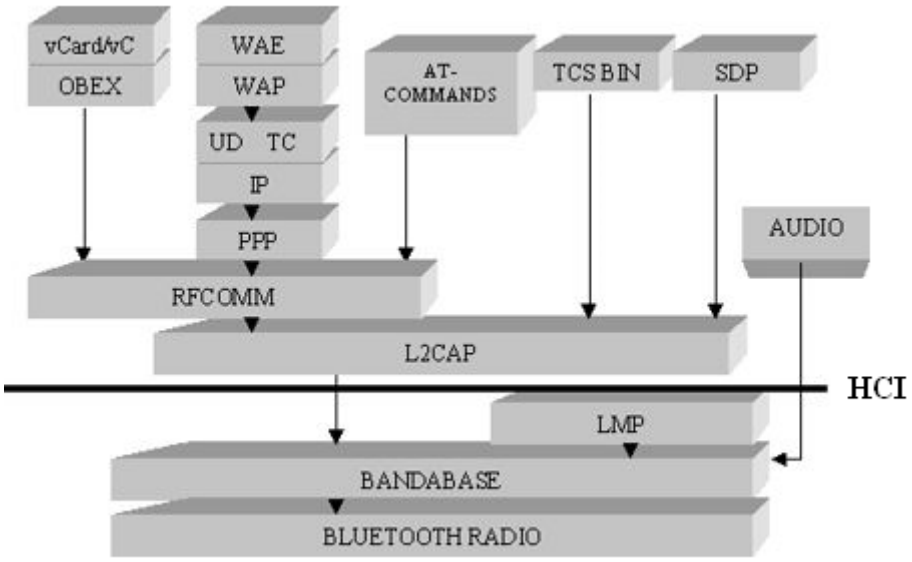


Figura 1. 2. *Pila de Protocolos Bluetooth*

La capa HCI separa el hardware del software y está parcialmente implementado en software y hardware/firmware. Los niveles inferiores al HCI están normalmente implementados en hardware y los niveles superiores al HCI en software.

A continuación se explican de forma más detallada las capas más significativas de la pila de protocolos.

Radio

Esta capa se encarga de especificar los detalles de la interfaz del aire como, por ejemplo, las ondas de frecuencia de los canales, los saltos en frecuencias, el esquema de modularización y los niveles de potencia de transmisión.

Banda Base

Se encarga del establecimiento de las conexiones en una piconet, además del formato de los paquetes, la temporización, control de la potencia y de las comunicaciones síncronas/asíncronas entre los pares de dispositivos.

Gestor del protocolo de enlace (LMP)

Es el responsable del establecimiento de los enlaces entre los dispositivos Bluetooth, además de la administración del mismo. Este nivel tiene también la misión de especificar y gestionar los establecimientos y cierres de conexiones, con todos los pasos que ello implica. El gestor de protocolo de enlace se encarga también de aspectos de seguridad (autenticación y encriptación) además del control y negociación del tamaño de los paquetes que se envían a Banda Base.

HCI (Controlador de interfaz)

Esta capa actúa como frontera entre las capas inferiores de la pila de protocolos Bluetooth y las capas superiores. La especificación Bluetooth define un estándar HCI que apoya a los sistemas que se ejecutan en dos procesadores separados. Por ejemplo, un sistema Bluetooth en un ordenador puede usar un módulo del procesador Bluetooth para implementar los niveles más bajos de la pila; se podría usar entonces el propio procesador del ordenador para implementar las capas superiores.

L2CAP (Logical Link Control and Adaptation Protocol)

Adapta los protocolos de la capa superior a la capa de banda base y provee servicios orientados a conexión y sin conexión. Desde esta capa hacia las superiores se implementa en software.

SDP (Service Discovery Protocol)

Permite a cualquier dispositivo Bluetooth encontrar dispositivos que se encuentren en su radio de alcance además de la búsqueda de servicios (y de sus características, parámetros/atributos). La búsqueda de servicios será un punto muy importante en el desarrollo del proyecto.

RFCOMM

Este nivel de la pila de protocolos representa a un puerto serie virtual y está diseñado para que la comunicación vía radio sea igual de simple que una comunicación puerto serie. Emula las señales del cable de conexión EIA-232 sobre la capa de Banda Base.

Las 2 capas siguientes no se usan en nuestro sistema, pero consideramos importante aportar una breve explicación sobre ellas.

TCS (Telephone Control Specification)

Este nivel define la señalización para el control de llamadas (para establecer llamadas de voz y datos entre dispositivos Bluetooth) y define los procedimientos de administración de movilidad para manejar grupos de dispositivos TCS.

OBEX (Object Exchange Protocol)

Este protocolo fue inicialmente desarrollado por IrDA (Infrared Data Association) para el intercambio de objetos por infrarrojos. OBEX suministra una funcionalidad similar al protocolo HTTP, pero de forma mucho más simple; además, proporciona un modelo para representar objetos y operaciones. Tal y como se ha explicado en los niveles de la pila de protocolos de Bluetooth (concretamente en el nivel L2CAP), la especificación define dos vínculos entre los dispositivos Bluetooth.

- Síncrono, orientado a conexión (SCO), para la comunicación de voz.
- Asíncrono, no orientado a conexión (ACL), para el envío de datos.

Cada tipo de vínculo se asocia con un tipo específico de paquetes.

Un enlace SCO ofrece el ancho de banda de canales reservados para la comunicación entre un maestro y un esclavo, soportando el envío de datos de forma regular sin la retransmisión de paquetes (retransmisión que podría producir retrasos, si de una comunicación por voz estamos hablando). En ACL existe un vínculo entre el maestro y el esclavo en el momento en que se realiza la conexión. Los paquetes de datos que usa Bluetooth en sus enlaces tienen todos 142 bits de codificación adicional, además de una carga que puede llegar a alcanzar los 2712 bits. La cantidad de codificación de los datos aumenta la seguridad en la transmisión. También ayuda a mantener un enlace de comunicaciones robusto en un entorno en el que hay otros dispositivos y mucho ruido.

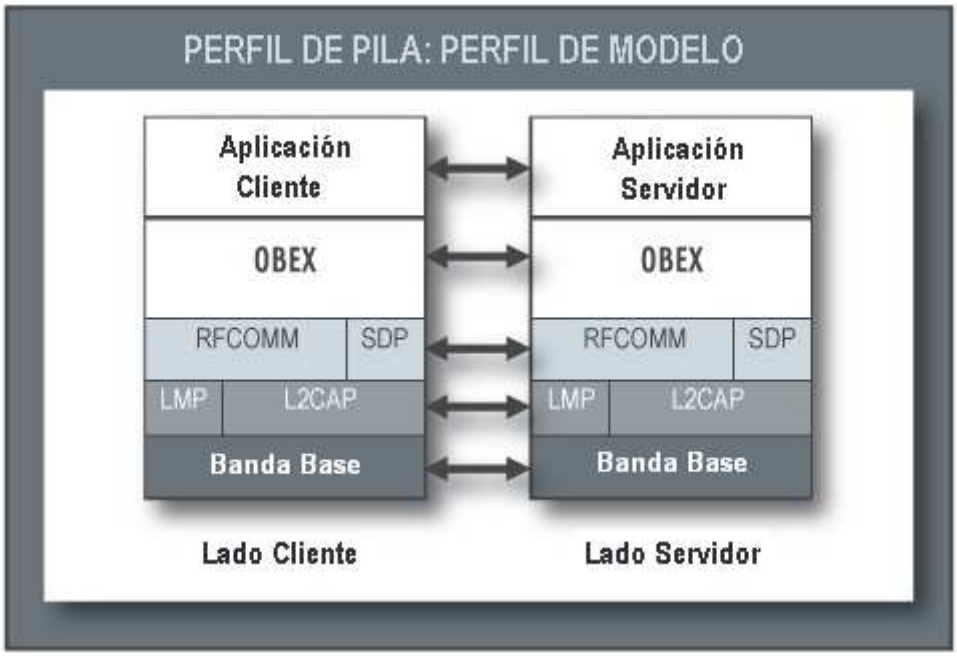


Figura 1. 3. *Modelo de comunicación entre protocolos*

1.8.2. Perfiles Bluetooth

Como explicábamos en el apartado de la historia del desarrollo de la especificación Bluetooth, ésta define una amplia gama de perfiles que a su vez definen una gran cantidad de tareas, algunas de las cuales no se han implementado todavía en ningún sistema. Siguiendo los procedimientos de los perfiles, los desarrolladores pueden tener por seguro que la aplicación que desarrollen funcionará con cualquier dispositivo que se ajuste a la especificación Bluetooth. Como mínimo, cada perfil contiene información sobre los siguientes temas:

- **Dependencias de otros perfiles.** Cada perfil depende de la base de perfil, llamado el perfil de acceso genérico, y algunas también dependen de los perfiles intermedios.

- **Sugerencias de formatos de interfaz de usuario.** Cada perfil describe cómo un usuario debería ver el perfil.

- **Determinadas partes de la pila de protocolos Bluetooth utilizados por el perfil.** Para realizar su tarea, cada perfil particular utiliza las opciones y los parámetros en cada una de las capas de la pila.

Los perfiles de Bluetooth están organizados en una jerarquía de grupos, con cada grupo dependiente de su predecesor. Mostramos a continuación los perfiles más importantes:

Figura 1.4. *Perfiles Bluetooth*

En la base de la jerarquía de perfiles tenemos el Perfil General de Acceso (GAP), el cual define un medio para establecer un enlace (en banda base) entre los dispositivos Bluetooth.

Además de esto, el GAP define:

- Características que deben aplicarse en todos los dispositivos Bluetooth.
- Procedimientos genéricos para descubrir y vincular a los dispositivos.
- Interfaz de usuario básica.

Todos los demás perfiles se basan en GAP; esto permite a cada uno de estos perfiles aprovecharse de las características que el GAP proporciona y asegura un alto grado de interoperabilidad entre las aplicaciones y los dispositivos. También hace más fácil a los desarrolladores definir nuevos perfiles a raíz de los actualmente definidos. A continuación pasamos a describir cada uno de los restantes perfiles.

- El perfil de **descubrimiento de servicios** describe cómo una aplicación debería usar el SDP (descrito en el apartado anterior) para describir servicios en un dispositivo remoto. Según lo estipulado en el GAP, cualquier dispositivo Bluetooth debería poder conectarse con cualquier otro dispositivo. Según esto, el perfil de descubrimiento de servicios requiere que cualquier aplicación pueda averiguar que servicios están disponibles en cualquier dispositivo Bluetooth, si nos conectamos a él.

- El perfil de **interfaz de usuario** (HID) describe cómo comunicar un dispositivo clase HID usando un enlace Bluetooth y cómo un dispositivo Bluetooth puede soportar servicios HID usando la capa L2CAP.

- Tal y como el nombre sugiere, el perfil de **puerto serie** define una emulación del cable serie RS-232 para dispositivos Bluetooth. Como tal, permite que el perfil de aplicaciones heredadas puedan usar Bluetooth como si fuese un enlace puerto serie. El perfil de puerto serie usa el protocolo RFCOMM para proporcionar la emulación del puerto serie.

- El perfil de Dial-up networking o de **acceso telefónico a redes** (DUN) está basado en el perfil de puerto serie y describe cómo una terminal de datos (ordenador portátil) puede utilizar un dispositivo de pasarela (un teléfono móvil o un módem) para acceder a la red telefónica. Al igual que otros perfiles, construidos sobre el perfil de puerto serie, el enlace virtual creado por los niveles inferiores de la pila de protocolos de Bluetooth es transparente a las aplicaciones usando el perfil DUN. Así, el controlador del módem en el terminal de datos no es consciente de que la comunicación se está haciendo a través de Bluetooth. Para la aplicación que funciona en el terminal de datos es igual de transparente la comunicación, ya que está conectada al dispositivo "pasarela" por un cable.

- El perfil de **auricular** (headset profile) describe cómo un auricular Bluetooth debe comunicarse con un ordenador u otro dispositivo Bluetooth (por ejemplo, un teléfono móvil).

- El perfil de **sustitución de cable** (HCRP) describe cómo enviar datos a través de un enlace Bluetooth a un dispositivo (una impresora). Aunque otros perfiles se pueden utilizar para la impresión, el HCRP está especialmente diseñado para aplicaciones que la apoyan. La compatibilidad entre dispositivos que empleen este protocolo depende de la presencia de aplicaciones paralelas que se ejecuten sobre la capa HCRP, tanto en el cliente como en el servidor. Para el cliente, esa aplicación será un controlador, mientras que en el dispositivo servidor será, por lo general, un intérprete o un generador de un formato de descripción de páginas.

- El perfil genérico de **intercambio de objetos**, proporciona un modelo genérico para otros perfiles usando el protocolo OBEX y define los papeles para los dispositivos clientes y servidores. Al igual que con todas las transacciones OBEX, el perfil genérico de intercambio de objetos estipula que el cliente inicia la transacción. El perfil, sin embargo, no describe cómo las aplicaciones deberían definir los objetos a intercambiar ni cómo se debería implementar exactamente el intercambio. Esos detalles se dejan en manos de los perfiles que dependen del perfil genérico de intercambio de objetos, es decir: del envío de objetos (Push), del intercambio de ficheros y de los perfiles de sincronización.

- El perfil de **carga de objetos** (Push) define las funciones de introducción de objetos tanto en el cliente como el en servidor. Estas funciones son análogas y deben interoperar con las funciones genéricas de intercambio de objetos del cliente y del servidor. El perfil de introducción de objetos se centra en una estrecha gama de formatos de objetos para maximizar la interoperabilidad. El formato más comúnmente aceptado es el formato vCard. En caso de que una aplicación quiera convertir información a otro formato, se deberá usar otro perfil, como el de transferencia de ficheros.

- El perfil de **transferencia de ficheros** también depende del perfil genérico de intercambio de objetos. Proporciona directrices para las aplicaciones que necesiten intercambio de ficheros y carpetas, en vez de los objetos más limitados que proporciona el perfil de introducción de objetos (visto en el párrafo anterior). Este perfil también define las funciones del cliente y del servidor y describe el rango de sus posibilidades en los diferentes escenarios.

- El perfil de **sincronización** es otro perfil dependiente del perfil genérico de intercambio de objetos. Describe cómo se pueden realizar aplicaciones de sincronización de datos entre, por ejemplo, una PDA y un ordenador. No es de extrañar que el perfil de sincronización también defina unas funciones de cliente y servidor. Este perfil se centra en el intercambio de información de gestión personal (PIM).

1.9. Seguridad en redes Bluetooth

A lo largo de este capítulo, hemos estado viendo cómo es posible la comunicación entre dispositivos Bluetooth o cómo poder transmitir datos (ya sea en forma de bits, fotos, textos...) de un lado a otro a través del aire. A esta comunicación se le puede conferir cierta seguridad.

Bluetooth ofrece tres **modos de seguridad**. El fabricante de cada producto determina el modo de seguridad del mismo.

Modo	Nombre	Descripción
1	No seguro	La seguridad no se aplica
2	Seguridad a nivel de servicio	Se concede el acceso a determinados servicios
3	Seguridad a nivel de enlace	La seguridad se aplica en un nivel común para todas las aplicaciones al inicio de la conexión

Tabla 1.1. *Modos de seguridad Bluetooth*

Los dispositivos y los servicios cuentan con distintos **niveles de seguridad**. Los niveles para los dispositivos son dos.

Nivel	Nombre	Descripción
1	Dispositivo de confianza	Ya ha sido emparejado con uno de sus dispositivos, tiene acceso sin restricción a todos los servicios
2	Dispositivo poco fiable	Dispositivo desconocido.

Tabla 1.2. *Niveles de seguridad para los dispositivos*

Los servicios cuentan con tres niveles de seguridad:

Nivel	Descripción
1	Servicios que precisan autorización y autenticación
2	Servicios que sólo precisan autenticación
3	Servicios abiertos a todos los dispositivos

Tabla 1.3. *Niveles de seguridad para los servicios*

En nuestro proyecto, las comunicaciones podrían ser abiertas y que para acceder a los servicios no hiciera falta ni autorización ni autenticación, pero hemos decidido mantener la autenticación del arduino por una mayor seguridad.

1.10. Bluetooth en Java

Mientras que el hardware Bluetooth había avanzado mucho, hasta hace relativamente poco no había manera de desarrollar aplicaciones Java Bluetooth. Fue el API **JSR-82** eñ que estandarizó la forma de desarrollar aplicaciones Bluetooth usando Java, de una manera en la que la complejidad del protocolo Bluetooth queda oculta tras el API, permitiendo al programador centrarse en el desarrollo en vez de en los detalles de bajo nivel del Bluetooth.

Aprovechando el carácter **multiplataforma** de Java, el objetivo de ésta especificación es definir un API estándar **abierto**, **no propietario** que pueda ser usado en todos los dispositivos que implementen **J2ME**. Por consiguiente fue diseñado usando, como explicaremos más adelante, los APIs J2ME y el entorno de trabajo CLDC/MIDP.

El JSR-82 es muy flexible, ya que permite trabajar tanto con aplicaciones nativas Bluetooth como con aplicaciones Java Bluetooth.

Está orientado a dispositivos que cumplan las siguientes características:

- Al menos 512K de memoria libre (ROM y RAM) (las aplicaciones necesitan memoria adicional).
- Conectividad a la red inalámbrica Bluetooth.
- Que tengan una implementación del J2ME CLDC.

E intenta ofrecer las siguientes capacidades:

- Registro de servicios.
- Descubrimiento de dispositivos y servicios.
- Establecer conexiones RFCOMM, L2CAP y OBEX entre dispositivos.
- Usar dichas conexiones para mandar y recibir datos (las comunicaciones de voz no están soportadas).
- Manejar y controlar las conexiones de comunicación.
- Ofrecer seguridad a dichas actividades.

Existen dispositivos móviles que soportan Java y tienen Bluetooth, pero sin embargo no soportan el API JSR-82. Esto quiere decir que no tenemos posibilidad de acceder al dispositivo Bluetooth a través de Java. Por ello habrá que acudir a las especificaciones del fabricante para cerciorarnos de que el API está soportado. A pesar de que el JSR-82 se especificó pensando en la plataforma J2ME, no sólo existen implementaciones y emuladores para éste. Debido a que J2ME es una versión reducida de J2SE, es perfectamente factible crear una implementación que pueda ser usada desde éste. De hecho existen emuladores e implementaciones, la mayoría son libres y suelen soportar dispositivos Bluetooth conectados al puerto serie. Ciertos emuladores y entornos de desarrollo también implementan este API simulando dispositivos Bluetooth, es decir, permiten realizar aplicaciones que usen el API JSR-82 sin necesidad de tener físicamente un dispositivo Bluetooth.

1.11. Bluetooth en la actualidad

A lo largo del 2007, el SIG anunció una nueva versión de la especificación Bluetooth: la versión 2.1 + EDR. El objetivo de esta nueva versión es facilitar a los consumidores la conexión con dispositivos Bluetooth. Esta transferencia de datos mejorada recibe el nombre de EDR, que es un método que amplía la capacidad y los tipos de paquetes Bluetooth para aumentar el rendimiento máximo, ofrecer una mayor compatibilidad para las conexiones múltiples y disminuir el consumo de energía, sin que se produzcan cambios en el resto de la arquitectura. Se puede establecer para que funcione de forma independiente en cada comunicación lógica. Una vez activada, los bits del paquete de la cabecera se interpretan de modo distinto a cuando operan en el modo de transferencia básico. Este cambio de interpretación se aprecia en el campo de la dirección de comunicación lógica de la cabecera. Como resultado, la cabecera y la carga útil se reciben y se modulan según el tipo de paquete.

Capítulo 2. J2SE & J2ME

2.1. Java

Alrededor de 1990, **James Gosling**, **Bill Joy** y otros desarrolladores de **Sun Microsystems**, empezaron a desarrollar un lenguaje de programación llamado **Oak**. La principal funcionalidad de este lenguaje era controlar los microprocesadores embebidos en aparatos electrónicos de consumo. Para ello, Oak necesitaba que fuese multi-plataforma (ya que estaban involucrados múltiples vendedores), extremadamente fiable y compacto

Sin embargo, en 1993, la televisión interactiva y el mercado de PDA no llegaron a despegar del todo. Fue entonces cuando llegó el éxito de Internet y la Web, así que Sun cambió rápidamente el objetivo de mercado, se centró en las aplicaciones de Internet y le cambió el nombre al proyecto; en ese momento fue cuando nació Java.

En 1994 apareció el navegador de Sun, **HotJava.** Escrito completamente en Java en solamente unos pocos meses, mostró el poder de los applets (programas que se ejecutan en el navegador) y también las habilidades de Java para el desarrollo de programas. A raíz del interés generado por Internet, el reconocimiento de Java creció rápidamente hasta convertirse en el software dominante para los navegadores y para las aplicaciones comerciales.

Aún así, las primeras versiones de Java no tenían la suficiente potencia ni capacidad para satisfacer las necesidades de una aplicación cliente. Por ejemplo, los gráficos de Java en la versión 1.0 eran demasiado burdos comparados con el software bien desarrollado y maduro de C y otros lenguajes.

Por otro lado, los **Applets** se hacían cada vez más populares. La capacidad de Java creció con el lanzamiento de una nueva versión expandida y se convirtió en un lenguaje muy popular para desarrollo empresarial, procesos de transacciones, interfaces con bases de datos, etc. Java se empezó a usar en pequeñas plataformas, como teléfonos móviles y PDAs. Actualmente, Java está en cientos de modelos de teléfonos.

2.2. J2ME (Java para móviles)

En 1998, Sun Microsystems volvió a centrarse en los dispositivos móviles, que eran los objetivos principales de Java cuando era conocido como Proyecto Oak. En 1998 Sun hizo público **PersonalJava** (también abreviado como pJava), con la intención de ser usado en los móviles y otros dispositivos limitados en recursos.

PersonalJava estaba basado en las clases del núcleo del **JDK1 1.8** e incluía algunas clases específicas para teléfonos móviles. De todas formas se comprobó que PersonalJava no tenía nada que hacer con dispositivos móviles bajos en recursos. De esta forma, en 1999 apareció **J2ME**.

En 2000 Sun extendió PersonalJava con el **API JavaPhone**, que encontró su lugar en muchas de las implementaciones del sistema operativo **Symbian**. Estaba orientado para PDA (Personal Digital Assistant) y SmartPhones. Muchas de estas mejoras se convirtieron en clases de los perfiles para J2ME

2.2.1. Máquinas virtuales de J2ME

Existen dos modelos de máquinas virtuales con las que se puede trabajar en J2ME: la **JVM** (Java Virtual Machine) y la **KVM** (Kuaui Virtual Machine). El modo de trabajo con una u otra no es una decisión que nosotros podamos tomar. Esta elección vendrá dada por las características básicas del dispositivo (memoria disponible y capacidad de proceso). Por otro lado, existen dos configuraciones básicas: **CDC** y **CLDC**.

El tipo de dispositivo móvil y, más concretamente, la máquina virtual que tenga implementada dicho dispositivo va a ser la responsable de que se utilice un tipo de configuración u otra. Así, si utilizamos un dispositivo que por sus características nos permita utilizar la JVM, entonces la configuración sobre la que podremos trabajar será la CDC y la máquina virtual la pasaremos a llamar **CVM** (Classic Virtual Machine) para denotar que se va a trabajar con esa configuración. Así mismo, si utilizamos la KVM, la configuración que deberemos trabajar es la CLDC. Dependiendo de las características del sistema operativo y del dispositivo en sí, se utiliza una máquina virtual y unas APIs propias de dicha máquina junto con las APIs propias del dispositivo. KVM es la máquina virtual de CLDC y está destinada a los dispositivos con pocos recursos. Está pensada para arquitecturas de 16bits y 32bits, pero con una memoria limitada que se mueva entre los 32kb y los 512kb, dependiendo del dispositivo.

Se diferencian dos **modos para la KVM**:

- El primer modo es apropiado para dispositivos, generalmente teléfonos móviles, que cuenten con unos 160Kb de memoria disponibles, ya que, para que funcione la KVM, se necesitan entre 40 y 80Kb y eso implica que como máximo se podrán ejecutar aplicaciones que tengan un tamaño de 12Kb. No obstante, a la hora de ejecutar una

aplicación hay que tener en cuenta otro factor y es que una aplicación va a necesitar 32Kb de memoria dinámica para su ejecución.
- El segundo modo se basa en una implementación más rica y está destinada a dispositivos de más de 160Kb y de hasta 512Kb.

Por su parte, JVM es la máquina virtual que también utiliza **J2SE** y está pensada para arquitecturas de 32 bits que además dispongan de una gran cantidad de memoria (entre 1Mb y 10Mb).

Por último, cabe destacar las **diferencias notables** que existen entre las dos máquinas virtuales que provocan que la KVM no tenga alguna de las cualidades de la JVM.

1) KVM no ofrece la posibilidad de trabajar con tipos de datos **float** ni **double** debido a que generalmente los dispositivos que utilizan KVM no precisan de cálculos de coma flotante.

2) No existen los llamados **class loaders** (clases que contienen el método main()) que un usuario podía definir. En su defecto se utilizan los predefinidos **starApp**.

3) No está presente el recolector de basura, por lo que hay que hacerlo manualmente.

4) Se elimina la interfaz nativa de Java.

5) Elimina los grupos de hilos e hilos demonios.

6) No existe un método que defina la finalización de clases.

7) Tiene una limitada capacidad para manejar excepciones.

8) Incluye una nueva biblioteca gráfica para ser empleada en dispositivos de poca memoria y con un tipo de pantalla más pequeña.

2.2.2. Configuraciones: CLDC

Recordemos que la configuración es un mínimo grupo de APIs útiles para desarrollar las aplicaciones destinadas a un amplio rango de dispositivos. Estas APIs tienen como función describir las características comunes de grupos de dispositivos como son:

- Características que son soportadas de Java.
- Características que soporta la Máquina Virtual Java correspondiente a las cualidades del dispositivo.
- Conjunto de bibliotecas básicas de Java y APIs soportadas.

J2ME posee dos configuraciones diferentes: CLDC (Conected Limited Device Configuration) y CDC (Conected Device Configuration). Cada tipo de configuración tiene un

objetivo distinto. CDC está concebida para dispositivos con más potencia de cálculo y de memoria. Está basada en J2SE e incorpora muchas de sus clases. En cambio, CLDC está concebida para dispositivos que tienen menos recursos.

En nuestro proyecto, ya que estamos trabajando con dispositivos móviles, usaremos la configuración CLDC, más concretamente la versión de **CLDC 1.1.** Los requisitos mínimos de hardware que contempla CLDC son:

- 160KB de memoria disponible para Java
- Procesador de 16 bits
- Consumo bajo de batería
- Conexión a red

Los dispositivos que claramente encajan dentro de este grupo son los teléfonos móviles, las PDA, los Pocket PC, etc.

En cuanto a los requisitos de memoria, según CLDC, los 160KB se utilizan de la siguiente forma:

- 128KB de memoria no volátil para la máquina virtual Java y para las librerías del API de CLDC
- 32KB de memoria volátil, para sistema de ejecución (Java Runtime System).

Nombre del Paquete CLDC	Descripción
Java.io	Clases y paquetes Standard de entrada/ salida
Java.lang	Clases e interfaces de la MV
Java.util	Clases, interfaces y utilidades estándar
Javax.microedition.io	Clases e interfaces de interconexión genérica CDDC

Tabla 2.1. *Paquetes característicos de CLDC*

2.2.3. Perfiles: MIDP

Podemos definir un perfil como un conjunto de APIs orientado a cubrir más específicamente las necesidades de funcionalidad de un grupo de dispositivos que comparten una serie de objetivos, por ejemplo, teléfonos móviles, PDA, electrodomésticos, etc. En la definición de perfil tiene gran importancia la interfaz gráfica, ya que nos podemos encontrar grandes diferencias, por ejemplo, entre la gran pantalla táctil de una PDA y la pantalla mucho más pequeña que presenta un teléfono móvil. Por ello, podemos concluir que dependiendo del tipo de configuración, nos encontraremos con distintos tipos de perfiles. Así tenemos:

- **Configuración CDC:**

  - Foundation Profile
  - Personal Profile
  - RMI Profile

- **Configuración CLDC:**

  - PDA Profile
  - Mobile Information Device Profile (MIDP)

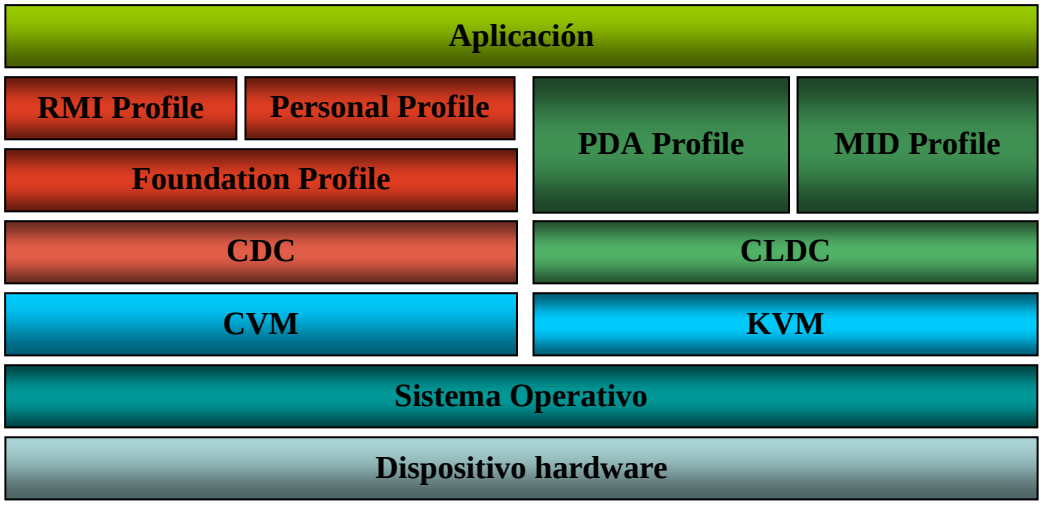


Figura 2.1. *Perfiles MIDP existentes*

CONFIGURACIÓN CDC

Foundation Profile

Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica como, por ejemplo, decodificadores de televisión digital.

Personal Profile

El Personal Profile es un subconjunto de la plataforma J2SE v1.3 y proporciona un entorno con un completo soporte gráfico AWT. El objetivo es dotar a la configuración CDC de una interfaz gráfica completa, con capacidades web y soporte de applets Java. Este perfil requiere una implementación del Foundation Profile.

RMI Profile

Este perfil requiere una implementación del Foundation Profile que se construye encima de él. El perfil RMI soporta un subconjunto de las APIs J2SE v1.3 RMI. Este perfil permite trabajar con RMI para acceder a métodos remotos.

CONFIGURACIÓN CLDC

PDA Profile

Está construido sobre CLDC. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 píxeles (al menos 200x100 píxeles) con un factor 2:1. Mobile Information Device Profile (MIDP): Este perfil está construido sobre la configuración.

MIDP

Al igual que CLDC fue la primera configuración definida para J2ME, MIDP fue el primer perfil definido para esta plataforma. Este perfil está orientado para dispositivos con reducida capacidad computacional y de memoria, conectividad limitada (en torno a 9600 bps), capacidad gráfica muy reducida (mínimo un display de 96x54 píxeles monocromo) y entrada de datos alfanumérica reducida. Los tipos de dispositivos que se adaptan a estas características son: teléfonos móviles, buscapersonas (pagers) o PDAs de gama baja con conectividad. Actualmente, este perfil tiene dos versiones, la última de las cuales proporciona APIs para los nuevos dispositivos, tales como los últimos teléfonos.

Vamos a centrarnos concretamente en el perfil **MIDP 2.0** ya que es el que vamos a usar a lo largo de todo el proyecto.

Este perfil está basado en la configuración CLDC, como ya hemos indicado, y los requisitos básicos son:

- Reducida capacidad de cómputo y memoria.
- Conectividad limitada.
- Ser capaz de interactuar con una mínima interfaz gráfica.
- Poseer una entrada de datos alfanumérica.
- 128 kb de memoria no volátil para componentes MIDP.
- 8 kb de memoria no volátil para poder escribir y leer datos de las aplicaciones que permanecerán en ella.

El perfil MIDP incluye una serie de paquetes en los cuales están definidas las clases e interfaces que podemos usar en una aplicación. Cada paquete está especializado en una labor concreta, por lo tanto, cada una de las clases e interfaces que contiene está pensado para llevar a cabo estas labores. En la tabla 2.2. podemos ver una breve descripción de la funcionalidad de cada uno de los paquetes que conforman el perfil MIDP.

Paquetes MIDP	Descripción
javax.microedition.lcdui	Clases e interfaces para GUI
javax.microedition.rms	Soporte para el almacenamiento persistente
javax.microedition.midlet	Clases de definición de la aplicación
javax.microedition.io	Clases e interfaces de conexión genérica
java.io	Clases y paquete estándar de E/S

java.lang	Clases e interfaces de la MV.
java.util	Clases, interfaces y utilidades estándar

Tabla 2.2. *Paquetes característicos de MIDP*

2.2.4. Modelo básico de una aplicación J2ME

En nuestro proyecto vamos a usar concretamente la configuración **CLDC** y el perfil **MIDP**. Al conjunto de aplicaciones que se basan en esta configuración y perfil se les denomina **MIDlet**. El objetivo final de la creación de una aplicación (MIDlet) en **J2ME** es la ejecución de ésta en un dispositivo, pero dicha ejecución implica una serie de pasos previos de los cuales se debe encargar el dispositivo en el cual se pretenda ejecutar. Las acciones que un dispositivo debe realizar antes de lanzar un MIDlet y que son condición necesaria para que funcione son:

- Poder localizar el MIDlet
- Poder descargar el MIDlet
- Proporcionar un medio para almacenar el MIDlet
- Gestionar el MIDlet

Para realizar estos pasos todo dispositivo que soporte CLDC y MIDP implementa un programa residente en memoria que es el encargado de hacerse cargo de todas estas tareas. Este programa se conoce como **Gestor de aplicaciones** o **AMS** (Application Management Software). En nuestro proyecto, crearemos los MIDlets y los enviaremos por Bluetooth al dispositivo (también podría enviarse por cable). El AMS será el encargado de controlar el proceso de descarga del MIDlet. El AMS, además de todo lo citado anteriormente, también se encargará del control del ciclo de vida del MIDlet, proceso que estudiaremos más adelante.

Por otra parte, hay que señalar que un dispositivo que soporte CLDC y MIDP también debe ser capaz de realizar otras funciones:

- Localizar los archivos JAD que pueden venir incluidos en un MIDlet.
- Ejecutar los MIDlets descargados.
- Conectarse con un servidor de internet utilizando HTTP y descargar tanto el MIDlet como su JAD.
- Instalar y borrar los MIDlets descargados.

Al crear un MIDlet tenemos que tener en cuenta que estamos trabajando para dispositivos de unas características muy limitadas. Eso implica que va a ser necesario realizar una serie de operaciones previas para que la máquina virtual de Java pueda interpretar y ejecutar la aplicación.

Por todo esto, tendremos que las fases que debemos seguir en el desarrollo de nuestro proyecto son tal y como aparecen en la figura que mostramos a continuación:

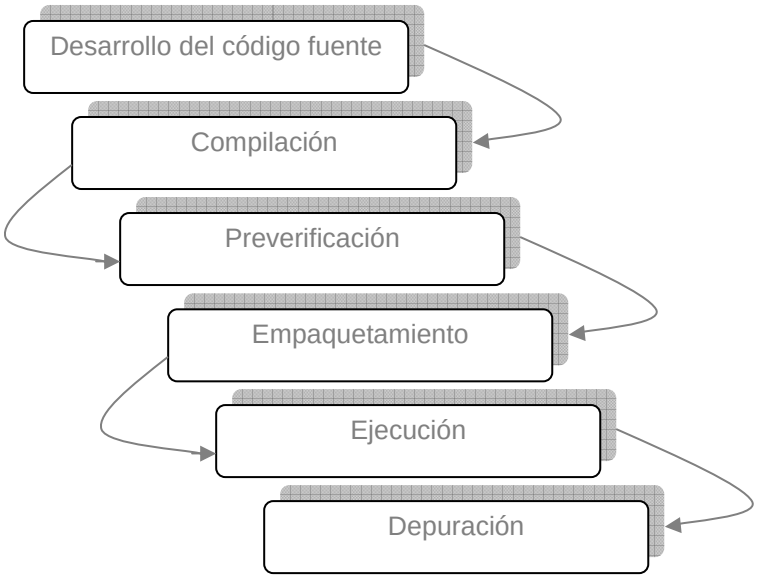


Figura 2.2. *Fases de desarrollo*

Desarrollo del código fuente

A la hora de crear una nueva aplicación, esta es la primera fase. Para crear un MIDlet, lo primero es escribir el código fuente de la aplicación. En nuestro caso usaremos el **Netbeans** como programa para realizar la codificación. Los detalles más reseñables a la hora del desarrollo del código los da la especificación del perfil MIDP. Como notas generales para un programador que ya conozca Java, indicar que MIDP no tiene una declaración de clases que lo inicialice (main()) sino que utiliza unos métodos concretos que son los que permiten comunicarse con el AMS (Application Management System). Estos métodos son:

- `startApp()`
- `destroyApp()`
- `pauseApp()`

Estructura de un MIDlet

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class MIDlet extends MIDlet {
      public void startApp() { }
      public void pauseApp() { }
      public void destroyApp(boolean unconditional) {   }

      }
```

Compilación

Una vez desarrollado nuestro código en un fichero .java, el siguiente paso es la compilación de dicho código. La compilación para un MIDlet sigue el mismo proceso que para crear cualquier otra aplicación en Java. El objetivo de efectuar la compilación es pasar de un lenguaje entendible por el hombre a un lenguaje que pueda ser entendido por la máquina virtual. Al compilar se van a generar un tipo de ficheros cuya extensión será .class. Los ficheros compilados (.class) están en formato bytecode y son entendibles por la KVM.

Preverificación

El preverificador de clases es una herramienta incorporada en la máquina virtual de J2SE y cuya utilidad es verificar los bytecodes de las clases Java. Tiene un peso bastante alto (hablando en Kbytes), por tanto, la implementación de un preverificador en la KVM es imposible ya que esta máquina virtual está pensada para poder ejecutarse en dispositivos de escasos recursos. Debido a este hecho, la KVM no incorpora ningún preverificador y se hace necesario efectuar esta operación previamente desde un entorno externo a la KVM.

Los preverificadores se encargarán de comprobar una serie de puntos:

- Que el código no sobrepase los límites de la pila de la máquina virtual.
- El uso de las variables y verificar que no se usan variables locales antes de ser inicializadas.
- Que se respete la estructura de las clases.

Empaquetamiento

Normalmente un MIDlet contiene múltiples ficheros; debido a esto hay que llevar a cabo un proceso para aglutinarlo todo en un único archivo para hacer así más fácil su distribución y descarga. El proceso de crear un único archivo se denomina empaquetamiento del MIDlet. El archivo que se genera al empaquetar los ficheros del MIDlet tiene extensión .jar. Opcionalmente se genera también un archivo .jad que es un archivo descriptor de la aplicación. El archivo JAR va a contener los ficheros de las clases que contienen el MIDlet, los recursos que utilice (por ejemplo, imágenes) y un fichero que describe el contenido del JAR que se denomina **archivo de manifiesto**. El archivo de manifiesto es simplemente un fichero de texto, pero con una estructura determinada. Se compone de una serie de atributos que son obligatorios y de otros que pueden ser omitidos.

Paquetes MIDP	Descripción
MIDlet-Name	Nombre de la MIDlet suite
MIDlet-Version	Versión de la MIDlet suite
MIDlet-Vendor	Creador del MIDlet
MIDlet-<n>	Contiene con el nombre e icono del MIDlet

Microedition- Configuration	Configuración necesaria para ejecutar el MIDlet
Microedition-Profiles	Perfil necesario para ejecutar el MIDlet

Tabla 2.3. *Atributos requeridos para un MIDlet*

Dependiendo de si utilizamos una herramienta de desarrollo o simplemente la línea de comandos, nos será más o menos engorroso crear el fichero JAR. En nuestro caso, usaremos Netbeans 6.0, que nos permite hacer todo esto de forma rápida y sencilla.

Ejecución y depuración

Para probar un MIDlet, tanto en funcionamiento como en depuración, lo más aconsejable es tener una herramienta de desarrollo, como ya hemos comentado antes; la herramienta que usaremos a lo largo del proyecto será **Netbeans 6.0**. Esta herramienta nos ofrece la posibilidad de ejecutar en simuladores nuestro MIDlet, con lo cual se evita el engorroso proceso de descargar el MIDlet en un dispositivo que soporte MIDP. También hay que tener en cuenta que el MIDlet puede tener errores y que al probarlo directamente podemos poner en peligro nuestro dispositivo. Por todo esto, es aconsejable que una vez creado el MIDlet, utilicemos un simulador para comprobar los posibles errores que pudiera tener.

2.2.5. Entorno de desarrollo: NetBeans

Esta herramienta desarrollada por SUN es un IDE para desarrollar todo tipo de aplicaciones en Java (J2EE, J2SE y J2ME). En anteriores versiones (anteriores a la 6.0) era necesario instalar un pack adicional para poder desarrollar aplicaciones J2ME (Mobility pack); actualmente, con la versión que hay hoy día, no es necesario instalar ese paquete. Así mismo, incluye todas las herramientas para el desarrollo de una aplicación, incluido un editor de textos en su interfaz que ayuda a la creación del código fuente de nuestra aplicación. En la página de NetBeans (http://www.netbeans.org) podremos encontrar toda la información necesaria sobre esta aplicación, además de poder descargarla de forma gratuita.

2.2.6. MIDlet

En este apartado vamos a analizar más profundamente qué caracteriza y cómo funciona un MIDlet. Como ya hemos visto antes, un MIDlet es una aplicación realizada en Java que se basa en la configuración CLDC y en el perfil MIDP. Está pensado para ser ejecutado en dispositivos inalámbricos móviles que se caracterizan por tener unos recursos limitados en cuanto a memoria, pantalla y capacidad de proceso.

Características

En la especificación MIDP encontramos el paquete javax.microedition, que tiene una clase que es la que da nombre a todas nuestras aplicaciones, la clase MIDlet. Para ejecutar un MIDlet, esta clase de alguna forma debe estar presente en el dispositivo. Para conseguir esta presencia continua, lo que hace el dispositivo es tenerla residente en parte de la memoria no volátil que éste posee. Esta memoria normalmente es de tipo ROM o EEPROM y está presente en todos los dispositivos. Un MIDlet se caracteriza también por ser una aplicación que debe ejecutarse con una orden del usuario y estar a la disposición de éste tantas veces como se quiera. Esta característica provoca que dicho MIDlet se deba encontrar almacenado en el dispositivo, en algún lugar físico para que, una vez descargado, el MIDlet pueda estar disponible en cualquier momento.

Gestor de aplicaciones o AMS

Por sí solo, un MIDlet no es capaz de ejecutarse ni de instalarse en el dispositivo. Tampoco nosotros, como usuarios, disponemos de un medio para realizar la instalación y posicionar el MIDlet en el lugar de almacenamiento. Para todas estas operaciones existe el llamado Gestor de aplicaciones o AMS (Application Management System). El AMS no es más que un software instalado y que está residente en cada dispositivo. Su misión es gestionar los MIDlets y realizar una serie de funciones para su control. Entre las funciones que realiza el AMS podemos destacar las siguientes:

- Gestionar el ciclo de vida del MIDlet, el cual se compone de 5 estados.
- Se encarga de controlar los estados por los que pasa el MIDlet.
- Gestiona la ejecución, destrucción y pausado del MIDlet.

2.2.6.1. Ciclo de vida

Todo MIDlet sigue un ciclo de vida que se compone de **5 fases**. Como hemos dicho antes, el encargado de controlar los estados de un MIDlet es el gestor de aplicaciones, que irá pasando por cada uno de ellos a medida que se hayan ido completando.

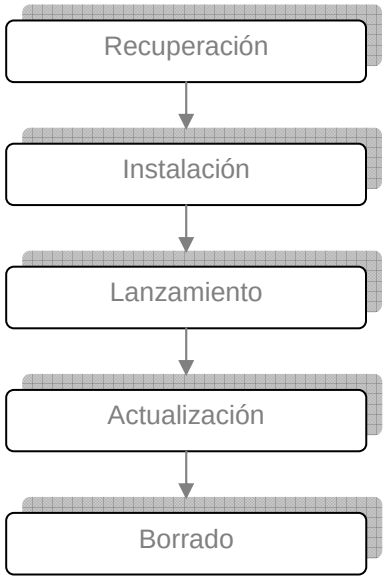


Figura 2.3. *Fases del ciclo de vida de un MIDlet*

Recuperación

La recuperación es la fase previa a la instalación del MIDlet. El proceso anterior necesario para llegar a la recuperación es el siguiente.

1. Usando el gestor de aplicaciones seleccionamos la aplicación que deseemos descargar. Previamente, a la hora de recuperar una aplicación, también debemos seleccionar el medio por el que se establece la conexión. Los medios de descarga son variados dependiendo de las características del dispositivo. Los más habituales son:
   - Descarga vía conexión web.
   - Descarga por un cable serie, directamente conectado al ordenador.
   - Descarga por redes inalámbricas (Bluetooth, en nuestro caso).
   - Descarga usando el puerto de infrarrojos.

2. Una vez localizado el MIDlet por el gestor de aplicaciones, se hace una serie de comprobaciones entre los dispositivos y el MIDlet para detectar los requisitos de este último (tamaño del MIDlet y cantidad de memoria que requiere).

3. Si el dispositivo tiene los suficientes recursos para gestionar el MIDlet, entonces éste es descargado y ubicado en la memoria del dispositivo.

Instalación

Cuando el gestor de aplicaciones ya ha descargado el MIDlet en el dispositivo, comienza el proceso de instalación. El propio AMS es el encargado de gestionar el proceso de instalación, así como los posibles errores o problemas que puedan surgir a lo largo de la misma. En el caso de que esto ocurra, se le notificará al usuario. Una vez instalado el MIDlet correctamente en el dispositivo, éste pasa a ser totalmente funcional y todas las clases, archivos de recursos y almacenamiento persistente que requiera pasan a estar preparados y listos para ser usados.

Lanzamiento

Cuando un usuario lanza un MIDlet, ya instalado en el dispositivo, éste entra en la KVM y comienza a ejecutarse. El gestor de aplicaciones tiene ahora la misión de gestionar los estados del MIDlet en función de los eventos que se produzcan durante dicha ejecución.

El proceso de lanzamiento de un MIDlet sigue este orden.

1. El AMS crea una nueva instancia del MIDlet llamando al constructor por defecto.

2. El MIDlet se coloca en estado de Pausa.

3. Cuando el usuario decide lanzarlo, el gestor de aplicaciones arranca el MIDlet poniéndolo en estado “activo” llamando al método starApp().

4. El MIDlet utiliza entonces cualquier recurso que necesite y que tenga a su disposición.

5. Si el AMS detecta alguna cosa anómala puede enviar la orden de pausa al MIDlet con la llamada al método pauseApp() o incluso puede llamar al método destroyApp() para destruirlo y descargarlo de la memoria.

6. En cualquier momento el MIDlet puede también hacer uso del almacenamiento y grabar su estado en una memoria no volátil para su posterior recuperación.

Actualización

Al descargar el MIDlet en un dispositivo puede darse el caso de que esta descarga sea una actualización de un MIDlet ya existente en el dispositivo. Al ser el gestor de aplicaciones el responsable de la instalación del MIDlet’s, éste debe ser capaz de detectar esta situación y proceder a la actualización del MIDlet correspondiente.

Cuando se descarga la actualización de un MIDlet existente, el AMS se comporta de la siguiente manera:

1. Comprueba que el MIDlet descargado es una actualización de uno ya existente.
2. Informa de esta situación al usuario.
3. Da la opción de efectuar la actualización o no.
4. Si la opción elegida es actualizar, realiza dicha actualización e informa al usuario de su finalización cuando ésta se produzca.
5. Si la opción elegida ha sido no actualizar, descarta el MIDlet que había sido descargado.

La forma que tiene el AMS de saber si un MIDlet es una actualización de otro es consultar el fichero de descripción y/o el fichero de manifiesto incluido en el JAR. Los atributos del MIDlet se encuentran en estos ficheros por lo que al consultarlos, el AMS puede conocer, entre otro muchos datos, el número de versión y la identificación del MIDlet.

Borrado

El gestor de aplicaciones es el responsable de su borrado total del dispositivo, siempre a petición del usuario. El gestor de aplicaciones pedirá siempre al usuario una confirmación de esta acción y una vez aceptada es cuando se realiza el borrado. El borrado no implica sólo quitar las clases y los archivos de recursos de un MIDlet instalado, sino que conlleva eliminar la información almacenada por ese MIDlet en el almacenamiento persistente.

2.2.6.2. Estados de los MIDlets

Un MIDlet, al estar cargado en memoria de ejecución, puede presentar en tres estados:

- Pausado
- Activado
- Destruido

Antes de considerar los distintos estados, lo primero que hay que indicar es que la memoria de ejecución es la memoria volátil que poseen los dispositivos para ejecutar aplicaciones. Es decir, que los tres estados que puede presentar un MIDlet sólo serán accesibles cuando el MIDlet esté ejecutándose. El encargado de gestionar los estados por los que puede pasar el MIDlet durante su ejecución es también el gestor de aplicaciones. El gestor de aplicaciones cambia el estado del MIDlet cuando se produce algún hecho

eventual, como, por ejemplo, que se reciba una llamada. En este momento, el AMS se encarga de activar el estado "Pausa" del MIDlet para que se pueda atender dicha llamada. Una vez terminada la llamada, el gestor de aplicaciones permite la opción de reanudar la aplicación volviéndola a pasar a "activo". Por último, cuando ya se termina de trabajar con el MIDlet y el usuario manda la orden de salir de él, el estado del MIDlet pasará a "Destruido". En ese instante el gestor de aplicaciones tiene la labor de eliminarlo de la memoria de ejecución del dispositivo.

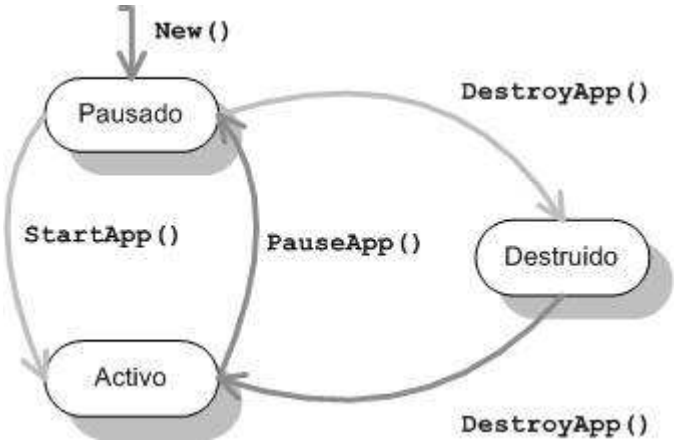


Figura 2.4. *Estados de los MIDlets*

2.2.7. Desarrollo de aplicaciones de alto nivel

MIDP nos provee de APIs para poder crear una aplicación de dos formas distintas:

- Usando una interfaz a "bajo nivel".
- Usando una interfaz a "alto nivel".

Una aplicación de **bajo nivel** se caracteriza porque para crearla se utiliza la pantalla como un lienzo en el que se tienen que ir dibujando píxel a píxel todos los elementos que la van a formar. Esta forma de trabajar es más pesada, pero consigue efectos visuales más logrados. Las aplicaciones de "bajo nivel" son ideales para el desarrollo de juegos y aplicaciones gráficas, ya que se obtiene un control total de todos los elementos de una pantalla.

Una aplicación de **alto nivel** se caracteriza porque para su creación se utilizan componentes definidos en MIDP y que cualquier dispositivo MIDP soporta. Estos componentes son cajas de texto, grupos de opciones, listas, etc. Estos componentes arriba mencionados tienen la desventaja de que no son personalizables, ya que cada dispositivo los implementa de una forma diferente, no obstante, tiene la gran ventaja de que son aplicaciones de un alto grado de estandarización.

En nuestro proyecto usaremos una interfaz de alto nivel, ya que se compone de una serie de menús, formularios y listas. Pero antes de estudiar más a fondo esta interfaz, sería

interesante hablar de algo fundamental: las clases que controlan el principal elemento de un dispositivo, es decir, la pantalla.

La **pantalla** es el medio por el cual el dispositivo se va a comunicar con el usuario para presentarle la información y gracias a ella el usuario va a poder interactuar también con el dispositivo en el momento en que éste solicite alguna acción. La clase ***Display*** es la encargada de controlar la pantalla. Toda aplicación que queramos crear va a tener un elemento común y este no va a ser otro que la clase Display, tanto si la aplicación es de "alto nivel" como de "bajo nivel".

Además de la clase Display, las aplicaciones que creemos van a poder hacer uso de una serie de clases generales muy útiles en el desarrollo de cualquier aplicación, como son:

- *Displayable*: representa cada una de las pantallas de la aplicación.
- *Command*: crea comandos generales o por pantallas para una aplicación.
- *CommandListener*: proporciona el medio para interpretar los comandos.

A continuación explicaremos brevemente cada una de estas clases.

Clase Displayable

La clase *Displayable* es una clase abstracta que va a representar cada una de las posibles pantallas que tenga nuestra aplicación. Un objeto de la clase *Display* va a ser capaz de tener varios objetos del tipo *Displayable*. Más adelante veremos algunos de esos objetos.

Clase Command

Los comandos nos proporcionan la posibilidad de interactuar con el MIDlet accionando una serie de funciones que todo dispositivo es capaz de ejecutar. La clase *Command* es la encargada de la creación y gestión de esos comandos.

Estos comandos pueden estar asociados a acciones tales como salir de la aplicación, volver a la pantalla anterior, enviar mensaje por Bluetooth (lo ajustamos a nuestra aplicación) o aceptar opción del menú.

Comandos	Acciones
`OK`	Confirma una selección
`CANCEL`	Cancela la acción actual
`BACK`	Traslada al usuario a la pantalla anterior
`STOP`	Detiene una operación
`HELP`	Muestra una ayuda
`SCREEN`	Tipo genérico para uso del programador referente a la pantalla actual
`ITEM`	Tipo genérico para uso del programador referente a un elemento de la pantalla actual

Tabla 2.5. *Acciones de los comandos de la clase Command*

Interfaz CommandListener

Un comando por sí solo no realiza ninguna acción, sino que hace falta un medio que nos permita detectar cuándo un usuario llama a ese comando. Para resolver este problema existen los **listeners u oyentes**. Un oyente o escuchador es un objeto cargado en memoria y cuya función es detectar eventos provocados habitualmente por el usuario como puede ser la detección de una tecla. La interfaz **CommandListener** nos proporciona el medio para crear estos oyentes.

Una vez vistas las clases genéricas para todo tipo de aplicaciones (ya sea de alto nivel o de bajo nivel), vamos ahora a adentrarnos un poco más en la interfaz de alto nivel ya que es la que vamos a usar en nuestra aplicación. Como vimos anteriormente, los componentes de las aplicaciones de alto nivel se caracterizan porque tienen ya un aspecto gráfico que viene predefinido por el dispositivo donde se vaya a ejecutar la aplicación y no se puede modificar. La ventaja de las aplicaciones de alto nivel es su portabilidad, ya que todo dispositivo que pueda ejecutar MIDP va a tener implementada la definición de componentes, aunque su aspecto sea distinto de un dispositivo a otro. Debido a la gran portabilidad de estas aplicaciones, el desarrollo usando APIs de alto nivel está enfocado al mercado de aplicaciones de negocio, las cuales no necesitan un aspecto gráfico detallado, sino que precisan de funcionalidad en cualquier dispositivo.

En las aplicaciones de alto nivel destaca la **clase Screen**, encargada de definir todos los elementos que formarán la pantalla de nuestra aplicación y la responsable de proporcionar lo métodos necesarios para la creación y manejo de nuestros componentes.

A continuación se van a explicar las clases más usadas para el desarrollo de nuestra aplicación.

2.2.7.1. Elementos de la interfaz

Ahora que tenemos una idea básica sobre el funcionamiento de un MIDlet, vamos a describir los elementos gráficos de los que disponemos para crear interfaces de usuario.

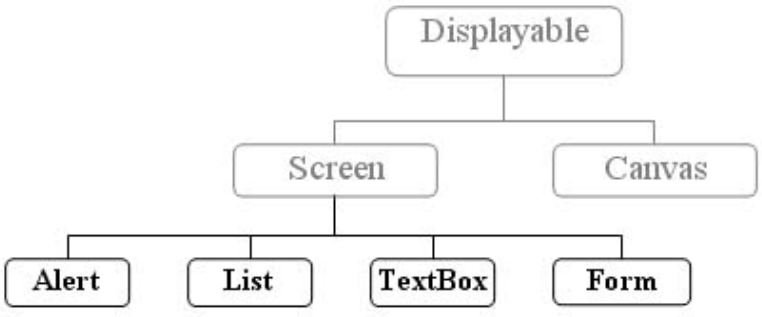


Figura 2.5. *Elementos de una interfaz de usuario*

Clase Alert

La clase Alert implementa una ventana que muestra un mensaje informativo al usuario. Su uso más habitual es informar al usuario que se ha producido un error en la aplicación o el aviso de algo importante. En nuestra aplicación usaremos las alertas para indicar, por ejemplo, que el dispositivo Bluetooth que estamos buscando no se ha encontrado. Esta ventana, mostrada mediante la clase Alert, puede permanecer por un tiempo predefinido y después desaparecer o permanecer en pantalla hasta que el usuario dé la orden de que desaparezca.

Clase List

La clase List proporciona un modo para crear una lista de elementos y suministra al usuario un medio para seleccionar los elementos que desee de dicha lista. En nuestro proyecto, las listas serán usadas para los posibles menús de la aplicación.

Clase TextBox

La clase TextBox nos permite editar o introducir texto. Aparece en modo pantalla completa, por lo que es muy parecido al TextArea de J2SE.

Clase Form

La clase Form nos permite crear formularios. Los formularios se caracterizan porque son capaces de contener otros objetos dentro de él y proporcionar un control de los eventos sobre esos objetos. Un objeto contenido dentro del formulario se denomina Item. Un ítem hereda de la clase Item y una de sus cualidades es que no ocupa toda la pantalla, como pasa con los objetos que proporcionan las clases anteriormente mencionadas (List, Alert...), sino que ocupan una parte. Como hemos dicho, un formulario puede contener varios Items, y cada ítem va a tener un índice asociado que es único y lo identifica en el formulario. El primer ítem añadido al formulario tendrá el índice 0 y seguidamente todos los índices del resto de los elementos irán aumentando de uno en uno. Los formularios son una parte fundamental de nuestra aplicación, se puede decir que el 90% de las pantallas de la aplicación consisten en formularios.

A continuación explicaremos brevemente cada uno de los elementos que pueden ser introducidos en el formulario:

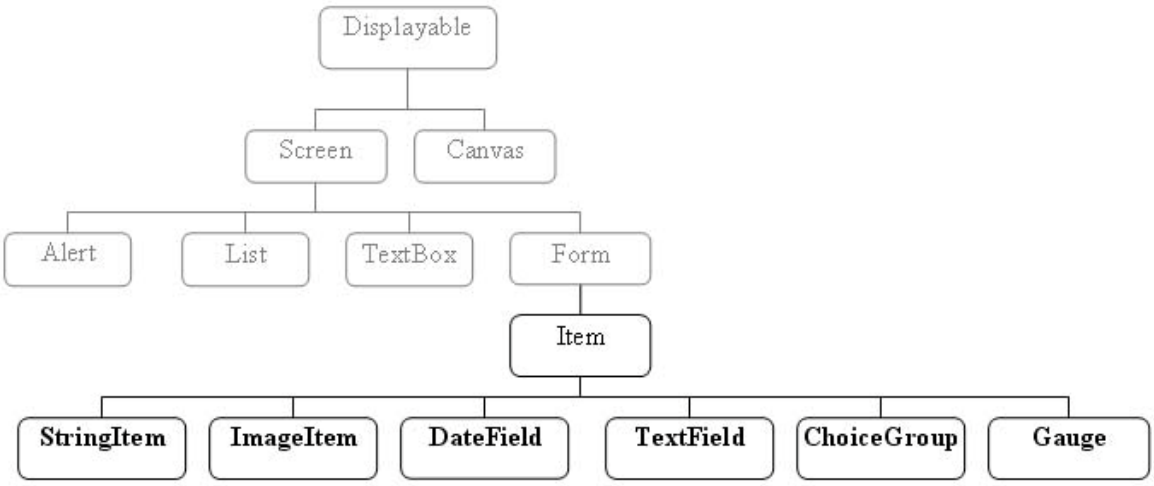


Figura 2.6. *Elementos de una interfaz de usuario. Formularios.*

`Clase StringItem`

La clase StringItem representa un objeto que tiene una cadena de texto y proporciona un medio para mostrar dicha cadena en un formulario. Podemos definir el estilo de esta cadena asignándole una fuente de la clase Font (de todas formas no hay muchas opciones disponibles). En nuestra aplicación, mediante los StringItems, mostraremos información relevante en los formularios

`Clase ImageItem`

La clase ImageItem posibilita la inclusión de imágenes dentro de un formulario.

`Clase DateField`

La clase DateField permite implementar un componente que nos sirva para controlar y definir fechas y horas de una manera más intuitiva. Este componente tiene la ventaja de que puede ser instalado en un formulario.

`Clase TextField`

La clase TextField nos proporciona una forma de insertar un campo de texto que nos posibilite introducir y editar texto dentro de un formulario.

`Clase ChoiceGroup`

La clase ChoiceGroup nos permite mostrar una lista de elementos en un formulario. Esta clase es muy similar a la clase List. Comparte con ella las propiedades de selección de

elementos de la lista y los métodos que se usan en dicha clase pero se diferencia en que ChoiceGroup está pensado para formar parte de un formulario.

`Clase Gauge`

La clase Gauge nos permite crear una barra de progreso a través de un gráfico de barras. Para nuestra aplicación, será útil para indicar al usuario que se está realizando una tarea (buscado dispositivos Bluetooth) y se debe esperar al resultado. Con esto, hemos dado un pequeño repaso a los elementos que van a componer la aplicación móvil. Más adelante, se verá cómo están dispuestos cada uno de los elementos en dicha aplicación.

2.2.8. Bluetooth en J2ME: JSR-82

Esta librería es fundamental para el desarrollo de nuestro proyecto ya que gracias a ella podremos establecer comunicación entre dispositivos mediante Bluetooth implementando código Java. Será usada en el desarrollo de la aplicación que irá alojada en el dispositivo móvil. Ya que esta API es la más importante del proyecto, vamos a dedicarle un poco más de tiempo para analizar cada uno de sus componentes. El API de Java para Bluetooth es un paquete definido por la Java Community Process (JSR-82). Este paquete proporciona una API para el desarrollo de aplicaciones Bluetooth.

A continuación mostramos la relación entre las APIs de Bluetooth y la plataforma J2ME, usando MIDP y CLDC (Explicado en los apartados anteriores).

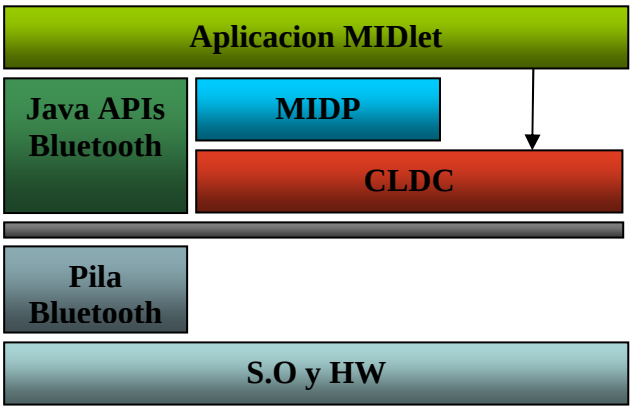


Figura 2.7. *APIs de Bluetooth-J2ME*

En la parte más baja de la pila nos encontramos con el hardware, el sistema operativo y la pila Bluetooth, seguida por la configuración (en nuestro caso CLDC), perfil (MIDP) y paquetes opcionales (en nuestro ejemplo, las APIs de Java para Bluetooth).

En la parte más alta, tenemos la aplicación (el MIDlet). La API de Java para Bluetooth define los siguientes paquetes:

Nombre del Paquete	Descripción
javax.microedition.io	El núcleo CLDC Generic Connection Framework
javax.bluetooth	El núcleo de la API Bluetoot
javax.obex	API de OBEX(OBject EXchange)

Tabla 2.6. *Paquetes de la API de Java para Bluetooth*

El paquete javax.bluetooth define clases e interfaces básicas para el descubrimiento de dispositivos, descubrimiento de servicios, conexión y comunicación. La comunicación a través de javax.bluetooth es a bajo nivel: mediante flujos de datos o mediante la transmisión de arrays de bytes. Por el contrario el paquete javax.obex permite manejar el protocolo de alto nivel OBEX y se usa generalmente para el intercambio de archivos.

A continuación detallaremos cada uno de estos paquetes, viendo su funcionalidad.

2.2.8.1. Paquete javax.bluetooth

Tal y como comentábamos en el apartado sobre Bluetooth, en toda comunicación de este tipo existe un dispositivo que ofrece un servicio (servidor) y otros dispositivos que acceden a él (clientes).

Dependiendo de qué partes de la comunicación queramos programar deberemos realizar una de las siguientes acciones:

Un **cliente Bluetooth** deberá:

1. Buscar dispositivos.
2. Buscar servicios.
3. Establecer la conexión.
4. Establecer la comunicación.

Por otro lado un **servidor Bluetooth** deberá hacer las siguientes operaciones:

1. Crear una conexión como servidor.
2. Especificar el servicio que va a ofrecer.
3. Abrir las conexiones de los clientes entrantes.

Antes de comentar cómo se ha de programar el dispositivo para que realice estas operaciones, vamos a detallar alguna de las clases básicas que nos harán falta a lo largo del desarrollo de esta explicación.

`Clase LocalDevice`

Un objeto de esta clase representa al dispositivo Bluetooth real. Este objeto es fundamental para cualquier operación que queramos realizar. A través de este objeto podemos obtener información sobre nuestro dispositivo Bluetooth, como, por ejemplo, su dirección o el nombre que lleva el dispositivo (friendly name).

`Clase RemoteDevice`

De la misma forma que LocalDevice nos da información sobre nuestro dispositivo local, RemoteDevice nos ofrece la misma información, pero esta vez de los dispositivos que hemos encontrado mediante búsquedas.

`Clase UUID`

Esta clase representa identificadores únicos universales (Universally unique identifier). Se trata de una serie de enteros de 128 bits que identifican protocolos y servicios. Mediante este UUID identificaremos el servicio que ofrecerá el servidor. Nuestra idea era crear un servicio propio para el Arduino, pero esto no fue posible debido a cierta limitación que comentaremos más adelante, por lo que hemos tenido que usar uno de los ya existentes.

A continuación explicaremos brevemente cada uno de los elementos necesarios para la búsqueda de dispositivos y servicios. Esta tarea sólo se realizará en los dispositivos clientes.

`Clase DiscoveryAgent`

Las búsquedas de dispositivos y servicios Bluetooth se hacen a través de un objeto de la clase DiscoveryAgent. A través de esta clase, tenemos la posibilidad de obtener un array de dispositivos Bluetooth ya conocidos por el usuario (PREKNOWN) o un array de dispositivos ya encontrados en búsquedas anteriores (CACHED). Las acciones anteriores no realizan ninguna búsqueda, sólo devuelve información de dispositivos remotos ya conocidos o encontrados en búsquedas anteriores.

Para iniciar una nueva búsqueda deberá llamarse al método `startInquiry()`. Este método recibe dos argumentos, el primero de ellos es un entero, que especifica el modo de conectividad que deben tener los dispositivos que vamos a buscar.

Este valor deberá ser:

- **`DiscoveryAgent.GIAC`** (General Inquiry Access Code, no hay límite en cuanto al tiempo que el dispositivo permanece en modo “descubierto”).

- **`DiscoveryAgent.LIAC`** (Limited Inquiry Access Code, el dispositivo sólo estará disponible por un periodo limitado de tiempo, después de pasado ese tiempo, el dispositivo pasa al modo oculto).

El segundo argumento del método `startInquiry()` es un objeto que implementa `DiscoveryListener`. A través de este último objeto serán notificados los dispositivos que se vayan descubriendo.

`Interfaz DiscoveryListener`

Esta interfaz es el elemento más importante a la hora de realizar búsquedas Bluetooth, ya que en él se definen todos los métodos necesarios para la búsqueda de dispositivos y servicios.

A continuación se detallan dichos métodos:

`public void deviceDiscovered(RemoteDevice rd, DeviceClass dc)`

Cada vez que un dispositivo es descubierto se llama a este método. Nos pasa dos argumentos: el primero de ellos es el objeto de la clase RemoteDevice que representa al dispositivo que se ha encontrado y el segundo argumento nos permitirá determinar el tipo de dispositivo encontrado.

`public void inquityCompleted(int c)`

Una vez que la búsqueda de dispositivos ha finalizado, se llama a este método. El argumento que se pasa nos indica el motivo de la finalización. Este argumento podrá tomar varios valores:

- **`DiscoveryListener.INQUIRY_COMPLETED:`** si la búsqueda ha concluido con normalidad.
- **`DiscoveryListener.INQUIRY_ERROR:`** cuando ha habido un error en el proceso de búsqueda.
- **`DiscoveryListener.INQUIRY_TERMINATED:`** si la búsqueda fue cancelada por el usuario.

Para realizar una búsqueda de servicios también usaremos la clase DiscoveryAgent e implementaremos la interfaz DiscoveryListener. En este caso los métodos a usar son los siguientes:

`public void searchServices(int[] atributos, UUID[] uuids, RemoteDevice rm, DiscoveryListener descubridor)`

Recurrimos a este método cuando queramos buscar servicios de un dispositivo en concreto. En el primer argumento le pasamos un array de enteros con los atributos de los servicios a los que estamos interesados, los servicios son descritos a través de los atributos que son identificados numéricamente. El segundo argumento es un array de identificadores de servicios que nos permite especificar los servicios en los que estamos interesados. El

tercer argumento es el dispositivo remoto sobre el que vamos a realizar la búsqueda de servicios. Por último pasaremos a un objeto que implemente la interfaz DiscoveryListener que será usada para notificar los eventos de búsqueda de servicios (al igual que la búsqueda de dispositivos).

`public void servicesDiscovered(int trId, ServiceRecord[] sr)`

Este método nos notifica que se han encontrado servicios el dispositivo buscado. El primer argumento es un entero que es el encargado de identificar el proceso de la búsqueda. El segundo argumento es un array de objetos ServiceRecord. Un objeto ServiceRecord describe las características de un servicio Bluetooth. Un servicio Bluetooth se describe mediante atributos, los cuales se identifican numéricamente, es decir, un servicio Bluetooth tiene una lista de pares identificador-valor que lo describen. El objeto ServiceRecord se encarga de almacenar esos pares. Estos identificadores son números enteros y los valores son objetos de tipo DataElement. Los objetos DataElement encapsulan los posibles tipos de datos mediante los cuales se pueden describir los servicios Bluetooth (enteros, arrays de bytes, UUIDs, booleanos, Strings...).

`public void serviceSerachCompleted (int trId, int rspCode)`

Este método es análogo al inquiryCompleted: cuando la búsqueda de servicios se ha completado se invoca este método. El primer argumento nos identifica el proceso de búsqueda. El segundo argumento indica el motivo de finalización de la búsqueda y puede tomar los siguientes valores:

- **`DiscoveryListener.SERVICE_SEARCH_COMPLETED:`** El proceso de búsqueda ha finalizado con normalidad.
- **`DiscoveryListener.SERVICE_SEARCH_TERMINATED:`** El proceso de búsqueda ha sido cancelado.
- **`DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:`** No se han encontrado registros en el proceso de búsqueda.
- **`DiscoveryListener.SERVICE_SEARCH_ERROR:`** Hubo un error durante el proceso de búsqueda.
- **`DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:`** No se pudo conectar al dispositivo sobre el que se quería realizar la búsqueda.

2.2.9. J2SE

Aunque lo que utilizamos en nuestro proyecto es J2ME, también tuvimos que hacer uso de J2SE para implementar un prototipo que simulara la función del Arduino. Puesto que aún no disponíamos del sistema empotrado, necesitábamos un servidor que se comunicara con el

cliente implementado con J2ME para verificar de la manera más sencilla y apropiada posible el correcto funcionamiento de la aplicación móvil y para ello se utilizó J2SE junto la librería Bluecove, bajo Windows.

Al inicio de este capítulo hemos comentado un poco el origen y la historia de Java. Vamos ahora a recordar los aspectos básicos de éste lenguaje

Java fue diseñado para ser:

- **Sencillo:** para que no requiera grandes esfuerzos de entrenamiento para los desarrolladores.
- **Orientado a objetos:** porque la tecnología de objetos se considera madura y es el enfoque más adecuado para las necesidades de los sistemas distribuidos y/o cliente/servidor.
- **Familiar:** porque su sintaxis está inspirada en la de C++ y se mantuvo lo más parecida a ésta, pero eliminando sus complejidades innecesarias.
- **Robusto:** simplificando la gestión de memoria y eliminando las complejidades de la gestión explicita de punteros y aritmética de punteros de C.
- **Seguro:** para que pueda operar en un entorno de red.
- **Independiente de la arquitectura:** Java está diseñado para soportar aplicaciones que serán instaladas en un entorno de red heterogéneo, con hardware y sistemas operativos diversos. Para hacer esto posible el compilador Java genera 'bytecodes', un formato de código independiente de la plataforma diseñado para transportar código eficientemente a través de múltiples plataformas de hardware y software.
- **Portable:** en el sentido de que es rigurosamente el mismo lenguaje en todas las plataformas. El 'bytecode' es traducido a código máquina y ejecutado por la Java Virtual Machine, que es la implementación Java para cada plataforma hardware-software concreta.
- **De alto rendimiento**: a pesar de ser interpretado, Java tiene en cuenta el rendimiento, y particularmente en las últimas versiones dispone de diversas herramientas para su optimización. Cuando se necesitan capacidades de proceso intensivas, pueden usarse llamadas a código nativo.
- **Interpretado, multi-hilo y dinámico**: El intérprete Java puede ejecutar bytecodes en cualquier máquina que disponga de una Máquina Virtual Java (JVM). Además Java incorpora capacidades avanzadas de ejecución multi-hilo (ejecución simultánea de más de un flujo de programa) y proporciona mecanismos de carga dinámica de clases en tiempo de ejecución.

J2SE (Java 2 Standard Edition) constituye una colección de APIs de Java, una plataforma orientada a entornos de gama media y estaciones de trabajo. Es decir, dirigida a usuarios como nosotros que queremos llevar a cabo la implementación de una aplicación con nuestro PC.

Incluye lo siguiente:

- Herramientas para generar programas Java. Compilador, depurador, herramienta para documentación, etc.
- La JVM, necesaria para ejecutar programas Java.
- La API de Java (jerarquía de clases).
- Código fuente de la API (Opcional).
- Documentación.

2.2.9.1. Swing

En las primeras versiones de la plataforma Java existían importantes limitaciones en las APIs de desarrollo gráfico (AWT). Desde la aparición de la librería Swing la situación mejoró sustancialmente y posteriormente, con la aparición de librerías como SWT, el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, etc. se vuelve relativamente sencillo.

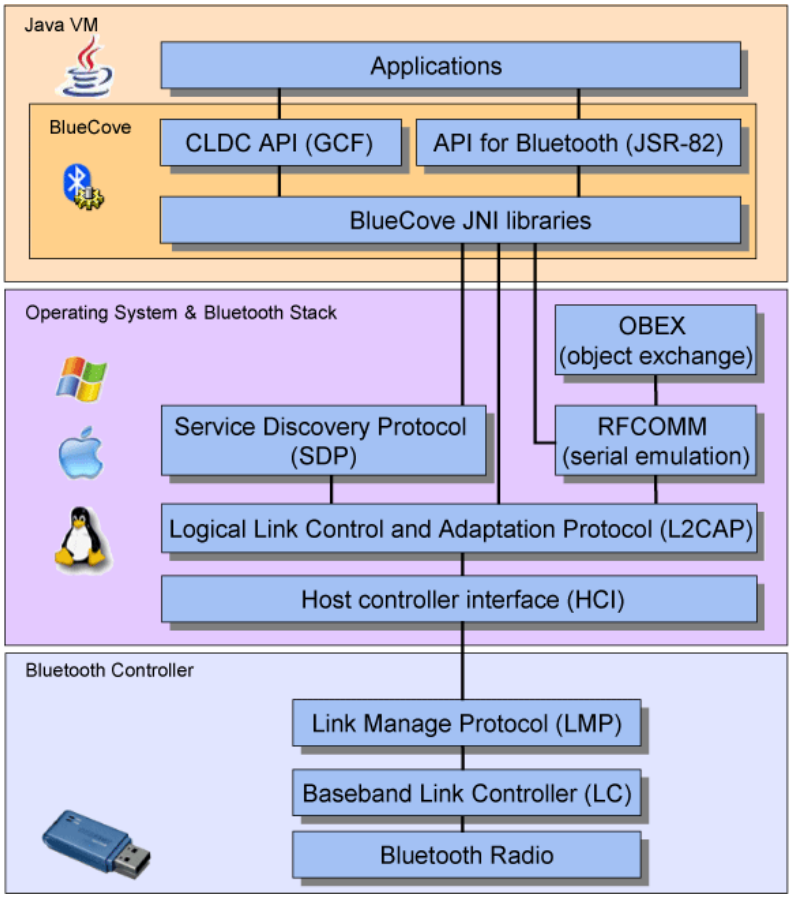
Swing es un widget toolkit para Java, es decir, un API que provee a los programadores de una interfaz gráfica. Proporciona componentes más sofisticados que los de la librería anterior AWT (Abstract Window Toolkit). Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

Ésta librería fue una de las principales a la hora de desarrollar nuestra aplicación, ya que tuvimos que implementar una interfaz sencilla que mostrara los datos necesarios para verificar que la conexión con el cliente móvil se estaba realizando correctamente. Esto se explicará de forma más detallada después, en el apartado que hemos dedicado a los prototipos.

Como ya hemos especificado en apartados anteriores, usamos el entorno Netbeans 6.0.

2.2.9.2. Bluetooth en J2SE: Bluecove

Ya hemos explicado que el API JSR-82 estandarizó la forma de desarrollar aplicaciones Bluetooth usando Java bajo J2ME. Por lo tanto, para que una aplicación en J2SE pueda también establecer conexión Bluetooth, es necesario emplear alguna herramienta válida para ello. Nosotros hemos utilizado Bluecove, una librería que implemente JSR-82 para J2SE y que soporta Mac OSX, WIDCOMM, BlueSolei, Windows nativo y Linux con BlueZ.


Java VM
Applications
BlueCove
CLDC API (GCF)
API for Bluetooth (JSR-82)
BlueCove JNI libraries
Operating System & Bluetooth Stack
OBEX
(object exchange)
Service Discovery Protocol
(SDP)
RFCOMM
(serial emulation)
Logical Link Control and Adaptation Protocol (L2CAP)
Host controller interface (HCI)
Bluetooth Controller
Link Manage Protocol (LMP)
Baseband Link Controller (LC)
Bluetooth Radio

Capítulo 3. LDAP

LDAP aparece con el estándar de los directorios de servicios. La versión original fue desarrollada por la Universidad de Michigan. La primera versión no se usó y fue en 1995 cuando se publicaron los RFC (Request For Comments) de la versión LDAPv2. Los RFC para la versión LDAPv3 fueron publicados en 1997. La versión 3 incluía características como las listas de acceso (control access lists) y replicación de directorios.

3.1. Tipo de protocolo

Las siglas LDAP vienen a significar *Lightweight Directory Acces Protocol*, por lo que nos sugiere un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido empleado para el almacenamiento y consulta de información en un entorno de red.

3.2. Directorio LDAP

Un directorio es un sistema de base de datos cuya característica principal es la de proporcionar una respuesta rápida a operaciones de búsquedas o consultas. LDAP funciona sobre Berkeley DB, un motor de base de datos transaccional y escalable que puede emplearse en cualquier aplicación y entre sus características destacan la sencillez, la eficiencia y la seguridad.

Utilizaremos para nuestro proyecto un LDAP sencillo que contenga información sobre los posibles usuarios del SAB y sus privilegios a la hora de acceder a unas habitaciones u otras. Este LDAP podría representar a pequeña escala el sistema de usuarios y accesos de una empresa o institución.

La estructura del directorio que emplearemos es la siguiente:

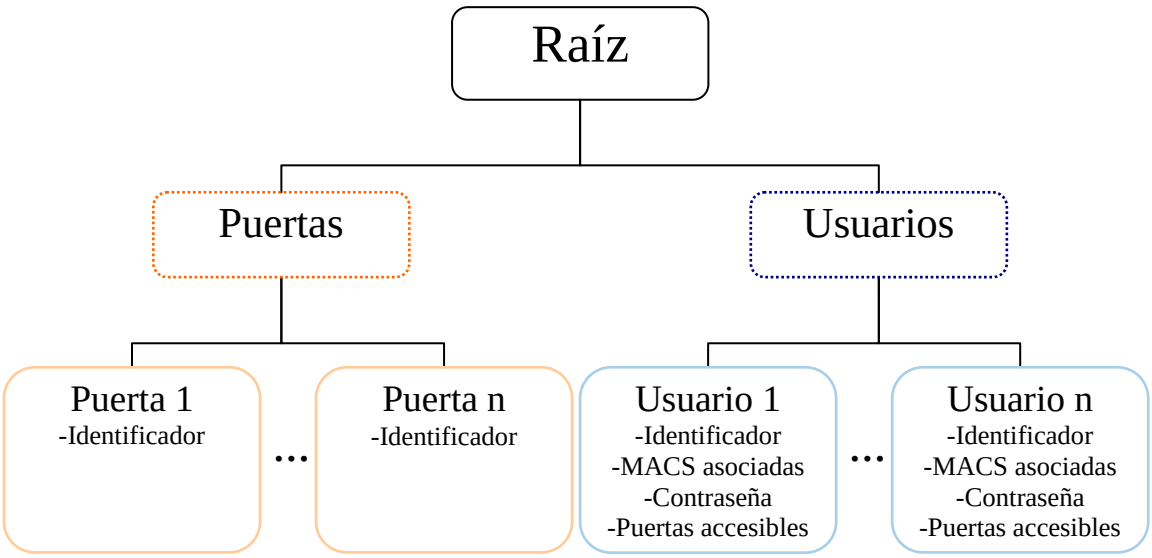


Figura 3.1. *Estructura directorio LDAP*

3.3. Operaciones

LDAP dispone de un conjunto de herramientas útiles que se pueden emplear para manipular información dentro de un directorio. Las citamos a continuación:

ldapSearch: Abre una conexión LDAP, enlaza y realiza una búsqueda acorde con los parámetros establecidos.

ldapCompare: Abre una conexión LDAP, enlaza y realiza una comparación acorde con los parámetros establecidos.

ldapAdd: Abre una conexión LDAP, enlaza y añade entradas al directorio.

ldapRemove: Abre una conexión LDAP, enlaza y elimina entradas del directorio.

ldapModify: Abre una conexión LDAP, enlaza y modifica entradas del directorio.

ldapModRDN: Abre una conexión LDAP, enlaza y modifica el RDN (Nombre distinguido raíz) de las entradas del directorio.

Capítulo 4. Sistema

4.1. Arquitectura

Los distintos elementos que forman el sistema de comunicación bidireccional son:

- Un dispositivo móvil.
- Un sistema empotrado.
- Una aplicación intermedia entre el sistema empotrado y el servidor LDAP.
- Un servidor LDAP.

Figura 4.1. *Comunicación entre los elementos del sistema*

El proceso de comunicación es el siguiente:

1. A través del dispositivo móvil conectamos mediante Bluetooth a un sistema empotrado asociado a una puerta en concreto. El dispositivo móvil detectará los identificadores de las puertas a las que puede tener acceso. Una vez seleccionada la

puerta, se le pasa al sistema empotrado la dirección mac del móvil y la contraseña del usuario encriptada para intentar acceder a dicha puerta, esta información se transmite con el formato: dirección_MAC#contraseña#$.

2. El sistema empotrado recibe una cadena correcta con el patrón dirección_MAC#contraseña#$ y cambia el caracter '$' por el número de puerta asociado al sistema empotrado. Esta nueva cadena es transmitida mediante UDP a la aplicación intermedia, que puede estar ejecutándose en un ordenador independiente o en el propio servidor LDAP.

3. La aplicación intermedia recibe por UDP una cadena con el formato dirección_MAC#contraseña#número_puerta, separa la cadena en distintas variables, desencripta la contraseña y realiza las consultas necesarias al servidor LDAP hasta averiguar si el usuario (identificado por su mac) con la contraseña especificada tiene acceso a la habitación controlada por el sistema empotrado.

4. El servidor LDAP responde las consultas de la aplicación intermedia.

5. La aplicación intermedia, una vez realizadas las consultas necesarias, responde mediante UDP al sistema empotrado indicándole si debe abrir la puerta o no.

6. El sistema empotrado abre o no la puerta en función de la respuesta que le ha enviado la aplicación intermedia y notifica el resultado al usuario mediante un mensaje en una pantalla LCD y mediante una comunicación con el dispositivo móvil.

4.2. Desarrollo

Como hemos especificado anteriormente, el sistema está formado por cuatro subsistemas, por lo que hemos tenido que llevar a cabo los siguientes módulos de desarrollo:

- Aplicación móvil.
- Aplicación de la placa empotrada.
- Aplicación intermedia.
- Servidor LDAP.

Cada una de estas aplicaciones ha sido programada en el lenguaje que más se ajusta a la tecnología a la que se aplica y para conseguir comunicación de datos entre ellas hemos utilizado los protocolos estándares.

A continuación se especifica una descripción de las implementaciones realizadas en cada caso y los problemas con los que nos hemos encontrado.

4.2.1. Dispositivo móvil

4.2.1.1. Aplicación móvil

La tecnología empleada en la mayoría de los entornos de dispositivos móviles es la llamada J2ME, en términos claros, Java para móviles. Por lo tanto, ésta es la que hemos elegido para programar la aplicación que irá en el dispositivo móvil en cuestión.

Comenzaremos pues explicando las distintas partes y funcionalidades que realiza la aplicación.

1. Al lanzar la aplicación nos aparecerá en nuestro dispositivo móvil la ventana principal con el logotipo de bienvenida y dos opciones a elegir: *Opciones* y *Salir*.

2. Al seleccionar *Opciones* se muestra un menú que proporciona 2 alternativas, *Entrar* y *Configurar.*

3. *Configurar* nos permite almacenar una contraseña o modificar/eliminar una ya almacenada.

4. Al seleccionar la opción *Entrar*, comenzará una búsqueda general que consiste en la detección de todos aquellos dispositivos Bluetooth (puertas) situados en un radio máximo de 150 metros (esta distancia dependerá del modelo de adaptador Bluetooth del que disponga el dispositivo móvil en cada caso).

5. Al finalizar dicha búsqueda nos aparecerá en pantalla una lista con todas las puertas a las que podemos tener acceso en principio.

6. El siguiente paso será seleccionar aquella puerta a la que deseemos entrar desbloqueando su sistema de cierre.

7. Una vez seleccionada la puerta elegida, nos aparecerá en la pantalla un campo de texto cifrado en el que tendremos que introducir nuestra contraseña y confirmar. Este campo no aparecerá si la contraseña está almacenada.

8. Transcurridos unos segundos obtendremos la respuesta que nos dirá si tenemos acceso a la puerta indicada con los datos proporcionados.

Se muestra a continuación un diagrama de flujo donde se pueden distinguir claramente los distintos caminos que puede tomar la aplicación.

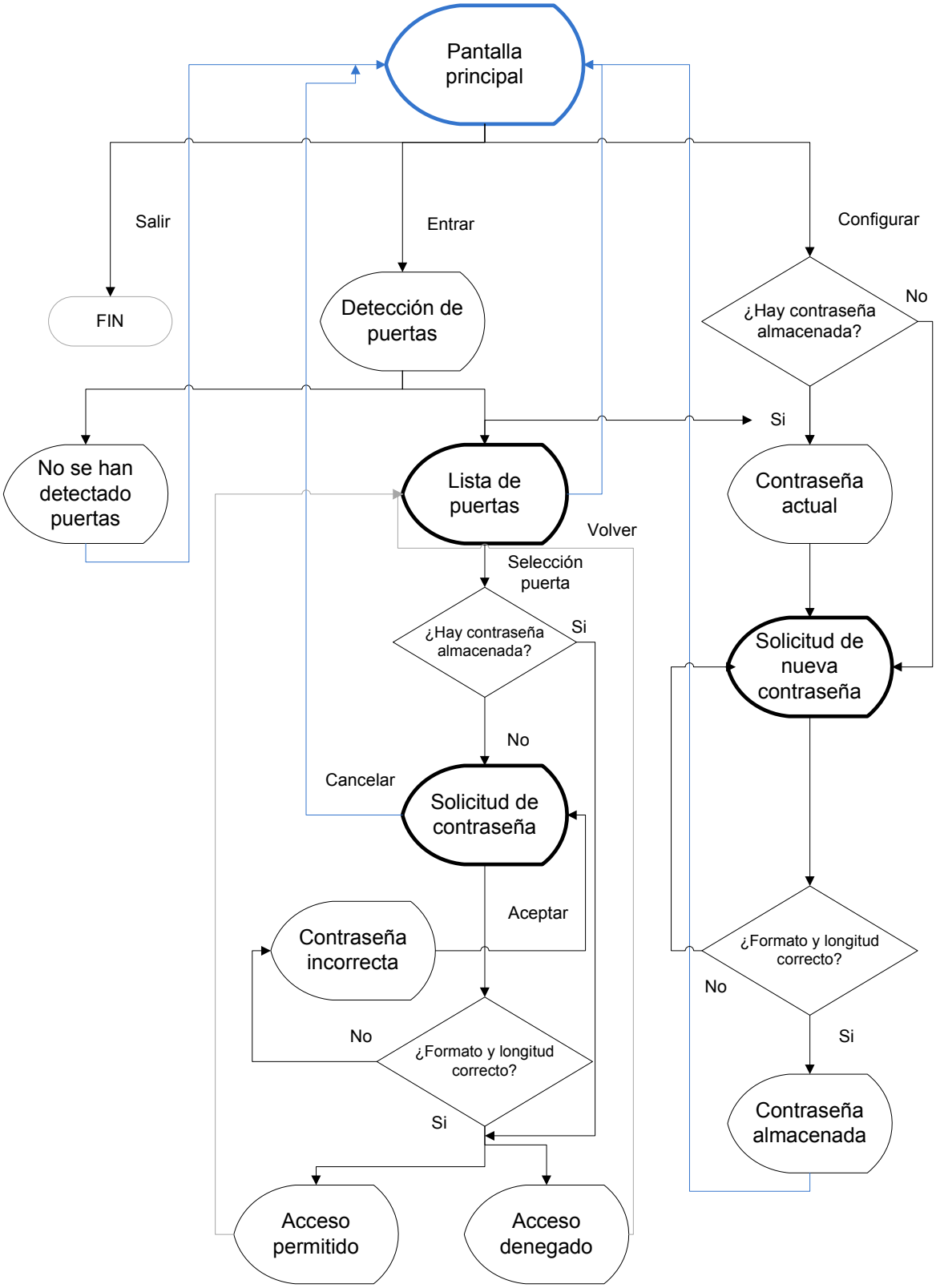


Figura 4.2. *Diagrama de flujo de la aplicación móvil*

4.2.1.2. Encriptación

Se ha utilizado el algoritmo de encriptación **AES** para cifrar la contraseña que se envía desde el dispositivo móvil, junto con la dirección MAC, a la aplicación intermedia encargada de llevar a cabo la desencriptación.

Concretamente, la contraseña se encripta con una clave de 16 caracteres, por lo que serán posibles contraseñas de hasta 15 caracteres, siendo esto fácilmente adaptable si las necesidades fueran otras.

Este algoritmo, AES (Advanced Encryption Estándar), ha sido elegido por ser suficientemente seguro y simétrico

4.2.1.3. Problemas encontrados

Para poder aplicar el algoritmo de encriptación que acabamos de explicar es necesario que los móviles dispongan de la **API JSR 177**. Por lo tanto, la opción para poder utilizar nuestra aplicación en todos los móviles es no implementar encriptación, pero esto no tiene sentido teniendo en cuenta que se trata de un sistema de control de acceso, en el que la seguridad es un factor fundamental. Debido al requerimiento de esta API, en este momento sólo los móviles de gama media-alta son compatibles, pero basándonos en la rapidez de desarrollo que se lleva acabo en el campo de la tecnología móvil, creemos que en un futuro todos lo serán. Por todo ello consideramos lógico y rentable asumir esta limitación. Los móviles que en la actualidad son compatibles con el SAB se pueden consultar en el anexo.

También tuvimos que considerar que el receptor final de la información era una aplicación C++ y no Java, por lo que hubo que estandarizar la comunicación para que fuera compatible entre ambas aplicaciones.

4.2.2. Sistema empotrado

4.2.2.1. Arquitectura hardware

Para el sistema empotrado hemos utilizado la plataforma **Arduino**. Los motivos por los que se eligió esta plataforma fueron varios: su facilidad de uso, su pequeño tamaño, su bajo coste y el hecho de ser de open-hardware, es decir, tanto su diseño como su distribución es libre y podemos, por tanto, utilizarla libremente para desarrollar nuestro proyecto sin tener que adquirir ningún tipo de licencia.

Arduino es una plataforma basada en una sencilla placa de entradas y salidas simple y un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring. Las plataformas Arduino están basadas en el microcontrolador ATmega168 o en el ATmega8, chips sencillos y de bajo coste que permite el desarrollo de múltiples diseños.

Nuestro sistema empotrado está formado por dos placas Arduino acopladas entre sí: una posee el módulo bluetooth y la otra permite la conexión ethernet.

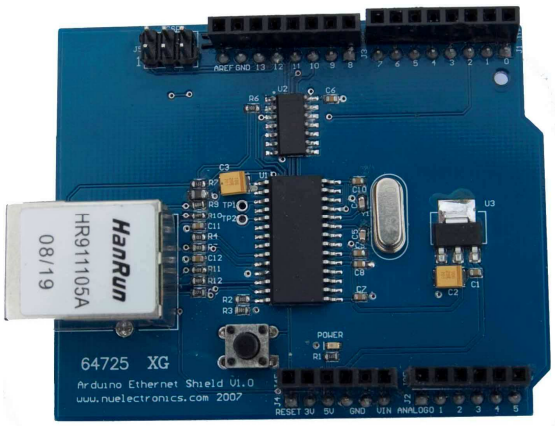


Figura 4.3. *Arduino Ethernet Shield*

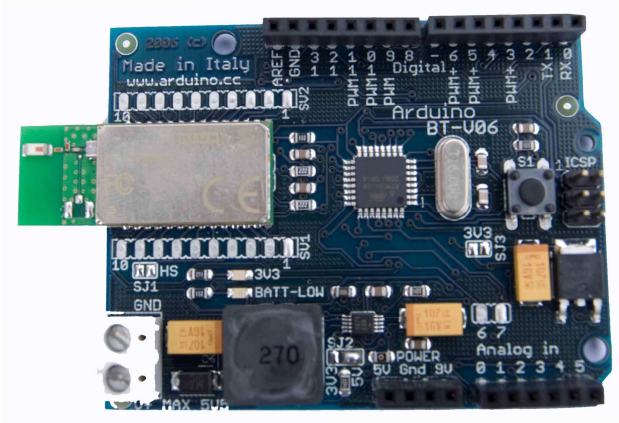


Figura 4.4. *Arduino Bluetooth*

El módulo bluetooth que posee la placa arduino es el bluegiga WT11. Éste es de clase 1, por lo que tiene un alcance aproximado de unos 100 metros. Por defecto viene con el protocolo iWrap integrado, un protocolo con el que simplemente es necesario enviar comandos AT a través del puerto serie del módulo para gestionar las conexiones. También es posible utilizarlo a través de **HCI** (Host Command Interface). Es decir, puede usarse simplemente como una interfaz radio a través de un puerto serie o USB (usando un procesador externo que cuente con la pila Bluetooth). Nosotros utilizamos el protocolo iWrap, lo que nos permite poner el logotipo Bluetooh en nuestros dispositivos finales sin

necesidad de pasar certificaciones adicionales, evitando el tiempo y dinero que eso conlleva.

Figura 4.5. *Bluegiga WT11*

Las placas Arduino llevan también acoplado un display LCD serie de 16x2, que además de conferir al sistema empotrado de un mejor acabado, cumple la función de mostrar mensajes de información dirigidos a los usuarios de SAB.

Este display tiene las siguientes características:

- Dos líneas (2 x 16) de caracteres.
- 5x8 puntos.
- Alimentación de +5V.
- Interfaces serie I2C, SPI o RS232/TTL (utilizamos el I2C).
- Modos del display: STN/Gray/Positive/Transflective.
- Alto rango de temperatura en el que es operativo: -20C - +70C.
- Driver: SPLC780D con PIC16F690. Proporciona funciones de alto nivel que pueden programarse sin necesidad de entender las instrucciones de control de bajo nivel del LCD.
- Brillo regulable con16 niveles.
- No necesita de ningún componente adicional para manejar el brillo de la pantalla.

Figura 4.6. *Display LCD*

4.2.2.2. Aplicación de la placa empotrada

El bucle principal que se está ejecutando en la placa empotrada funciona como una pequeña máquina de estados. Entre cada iteración del bucle hay un delay de 10 milisegundos para tener un control sobre el tiempo. Cada estado tiene funciones concretas:

Estado 1 - Espera/Lectura: este es el estado habitual del sistema, en él se espera una conexión entrante por bluetooth y se leen los caracteres recibidos hasta reconocer una cadena correcta. Cuando reconoce una cadena correcta pasa al siguiente estado.

Estado 2 - Envío UDP: en este estado se construye y envía a la aplicación intermedia el paquete UDP correspondiente a la cadena reconocida en el estado 1 y se inicializa un contador para controlar el tiempo de espera para recibir respuesta. Tras hacer esto pasa al estado 3.

Estado 3 - Recepción UDP/Reinicio: este estado queda a la espera de recibir respuesta de la aplicación intermedia mediante UDP. Cuando recibe la respuesta es el encargado de abrir la puerta si es necesario, mostrar el mensaje correspondiente en el LCD, enviar al dispositivo móvil el resultado de su intento de entrar y reiniciar las variables necesarias para el próximo intento de acceder a la habitación (se vuelve al estado 1). Además es el encargado de controlar las dos variables de timeout de la aplicación. La primera sirve para controlar el tiempo que ha pasado desde que se envió el paquete UDP a la aplicación intermedia, por lo que pasado un cierto límite de tiempo sin recibir respuesta se reenvía el paquete UDP para volver a intentar realizar la consulta (se vuelve al estado 2). La segunda controla el número de veces que se ha intentado enviar el UDP y si se supera el límite marcado por la aplicación se muestra un mensaje de error de red en el LCD y se reinicializan las variables (se vuelve al estado 1).

4.2.2.3. Problemas encontrados

La característica principal de este módulo Bluetooth es que facilita enormemente la capacidad de añadir a un diseño conectividad Bluetooth sin ningún tipo de complicación, pero por esto mismo nos hemos encontrado con una limitación: no es posible crear un servicio propio, por lo que nos vemos obligados a utilizar el servicio estándar 0x1101 para el puerto serie.

El problema que esto nos plantea es que al ser un servicio muy común, en la detección de dispositivos podrían encontrarse, además de los microprocesadores de cada puerta, cualquier otro dispositivo que estando al alcance tuviera ese servicio.

La solución que hemos adoptado para este problema consiste en establecer un rango de direcciones MAC para los sistemas empotrados, pudiendo de esta manera filtrar en la aplicación móvil los dispositivos encontrados, limitándonos a considerar aquellos cuya MAC se encuentre en el rango establecido.

Ha sido necesario eliminar de la librería que maneja la placa ethernet las funciones no utilizadas (como la comunicación TCP) debido al limitado espacio disponible en nuestro sistema empotrado.

Han surgido algunos problemas durante el desarrollo del código debido a que el entorno de desarrollo oficial para Arduino presenta numerosos bugs ya que aún es de reciente creación. Por esto, las posibilidades que ofrece el LCD en cuanto a dinamizar visualmente el sistema no han podido ser tan aprovechadas como nos habría gustado.

Durante una pruebas, a un par de meses de la finalización completa del proyecto, tuvimos un problema por un mal contacto y la placa Bluetooth se quemó. Por suerte se nos proporcionó una nueva placa y se pudo solucionar este percance con rapidez

4.2.3. Aplicación intermedia

4.2.3.1. Aplicación C++

Programa que tiene como función principal realizar consultas que validen los datos que le llegan a través de comunicación socket vía UDP desde la placa del sistema empotrado. Su comportamiento es similar a un demonio. Está corriendo continuamente bajo entorno Unix/Linux a la espera de conexiones de sockets entrantes. Está programado en C++ y hace uso de la librería ldapc++ desarrollada por Ralf Haferkamp para poder realizar consultas al servidor LDAP montado en la misma máquina en que se ejecuta o en una remota.

La aplicación se encuentra en un bucle infinito esperando mensajes UDP:

1. Inicialmente se encuentra a la espera de conexiones sockets entrantes.

2. Si detecta un socket entrante cuya secuencia de datos es reconocida, se procede a la separación de los datos en las distintas variables que utilizaremos al realizar las consultas.

3. Teniendo como argumentos las variables establecidas, se procede a realizar una petición de conexión y consulta al servidor LDAP, comprobando que los datos introducidos se corresponden con los que figuran en el directorio LDAP y que el usuario tiene acceso a la habitación pedida.

4. Una vez consultados los datos, el resultado de la validación se enviará en modo de respuesta mediante un socket de salida por el puerto correspondiente a la placa del sistema empotrado.

5. Se vuelve de nuevo al estado inicial del bucle de espera.

4.2.3.2. Problemas encontrados

La creación de esta aplicación intermedia fue en sí misma la solución a un problema, pues no se disponía de espacio suficiente en la memoria del sistema empotrado para poder implementar en éste estas funciones.

A parte de eso no encontramos problemas destacables más allá de los típicos que conlleva el tener que documentarnos sobre el uso de sockets para poder enviar y recibir mensajes, la librería ldapc++ para comunicarnos con el servidor LDAP y la librería openssl para descodificar la contraseña del usuario.

4.2.4. Servidor LDAP

4.2.4.1. Contenido del servidor

El contenido del directorio LDAP del SAB sigue la estructura indicada en el punto 3.2. Como se puede observar, dc=dacya, dc=ucm, dc=es es la raíz del árbol que conforma su estructura general.

Figura 4.7. *Estructura LDAP*

A continuación se muestra el contenido de la puerta 1. El resto de puertas son iguales, a excepción del nombre del identificador.

Figura 4.8. *Estructura de una puerta*

El contenido de un usuario es algo más complejo que el de una puerta. Son 3 los elementos que se requieren como parte de su estructura. Un identificador, en el caso del ejemplo "becario". Las direcciones MAC asociadas, es decir, las MAC de los dispositivos móviles que ese usuario tiene registrados como llaves de acceso a las estancias, lo normal es disponer sólo de una dirección registrada, pero como se observa en el ejemplo es posible tener más de una. Por último, el conjunto de puertas que el usuario tiene autorización para abrir, en el caso del ejemplo la 1 y la 3.

Figura 4.8. *Estructura de un usuario*

4.2.4.2. Problemas encontrados

El único problema encontrado fue el absoluto desconocimiento de la tecnología LDAP. Ya que conocer a fondo esta tecnología no era uno de los principales objetivos del proyecto, decidimos utilizar la aplicación **phpLDAPadmin** como interfaz para facilitar la creación e inserción de la información en nuestro servidor LDAP.

4.3. Funcionamiento y evolución del prototipo

El esquema del sistema tal y como lo tenemos nosotros en funcionamiento es el siguiente:

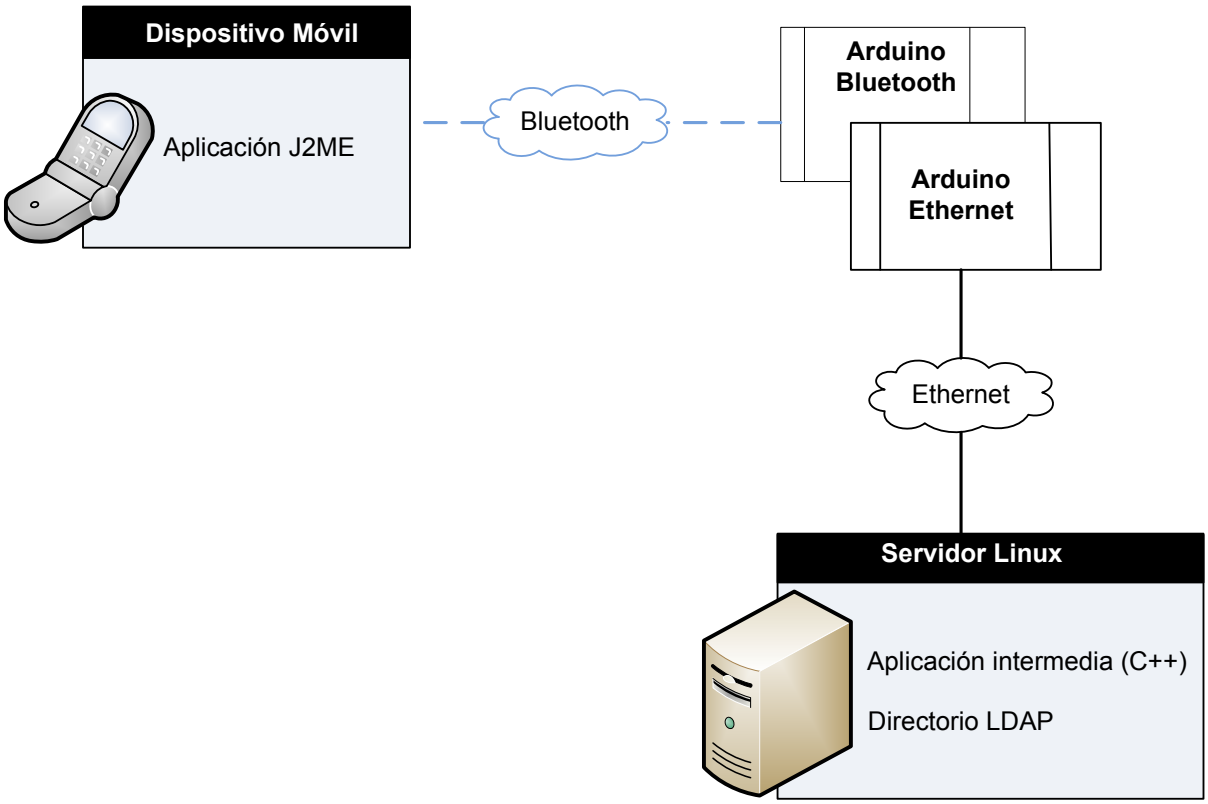


Figura 4.9. *Esquema del sistema*

En un principio no disponíamos de las placas de Arduino, por lo que como ya comentamos en el apartado 2.2.9, tuvimos que hacer uso de J2SE para implementar una aplicación que simulara la función de éstas. Un servidor que se comunicara con la aplicación móvil y con la aplicación intermedia. Este **Servidor BT** que forma parte de la primera versión funcional del prototipo consta de una interfaz sencilla que permite especificar el número de puerta que simula ser, la IP de la máquina en la que se encuentra el **ServidorSk** (aplicación

intermedia) esperando las conexiones y el puerto por el cual se le envía a ésta la información recibida del cliente móvil. Para poder implementar este servidor BT en Java y bajo Windows tuvimos que hacer uso de la librería **Bluecove**.

Se puede ver en las imágenes siguientes un ejemplo de ejecución de esta primera versión. Aún no estaba implementada la encriptación, es decir, la contraseña se envía desde el dispositivo móvil sin cifrar e igualmente se trata así en la aplicación intermedia.

Figura 4.10. *Funcionamiento prototipo*

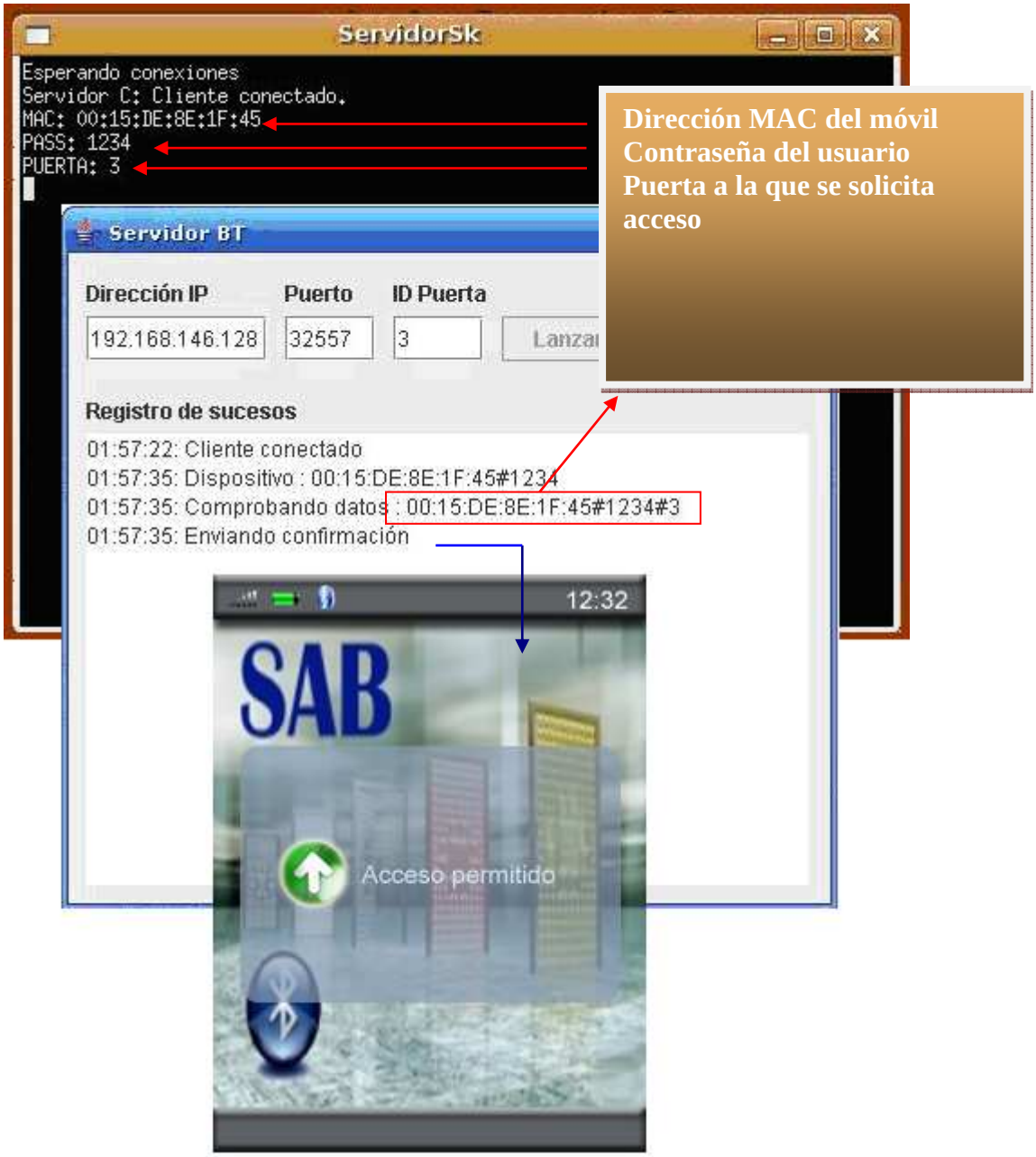


Figura 4.11. *Funcionamiento prototipo*

Se pasó por varias fases de ubicación de las aplicaciones.

1. Una primera en la que todo se encontraba en la misma máquina y las conexiones se realizaban en local:

   - Máquina principal con Windows:
     - Servidor BT implementado con J2SE.

- Máquina virtual dentro de la principal, con Linux:
  - ServidorSk (aplicación intermedia).
  - Servidor LDAP.

2. Una segunda en la que el directorio LDAP se encontraba en una máquina externa, de otra red.

3. Una tercera en la que también la aplicación intermedia se encontraba en un PC de otra red.

4. Una última en la que tanto el directorio LDAP como la aplicación intermedia se albergaron en un servidor externo con Linux.

Después de pasar por diferentes fases y mejoras, obtuvimos la siguiente versión del prototipo cuyas ampliaciones a destacar son:

- Sustitución del Servidor BT por el sistema HW constituido por las placas Arduino (placa Ethernet y placa Bluetooth).

- Encriptación de los datos enviados por el dispositivo móvil, con los consiguientes cambios realizados en la aplicación intermedia para poder tratar la información cifrada.

- Posibilidad de guardar la contraseña en la aplicación móvil.

A continuación presentamos imágenes de las placas Arduino y de la interfaz de la aplicación móvil.

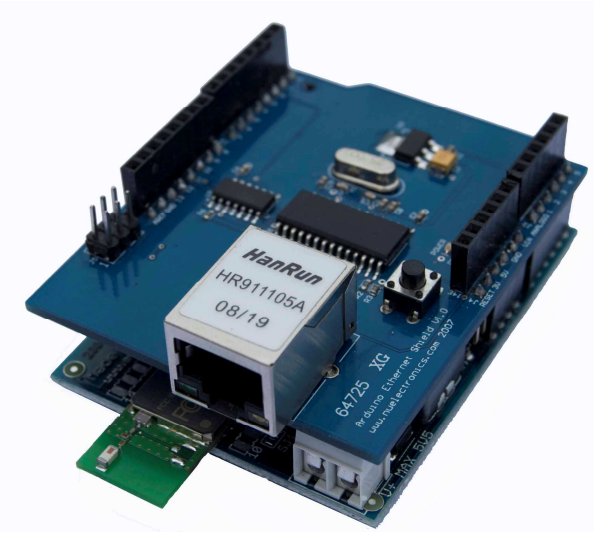


Figura 4.12. *Placas Arduino del prototipo (Ethernet arriba, Bluetooth abajo)*

Solicitud de acceso

Secuencia de las pantallas de la aplicación móvil en una solicitud de acceso.

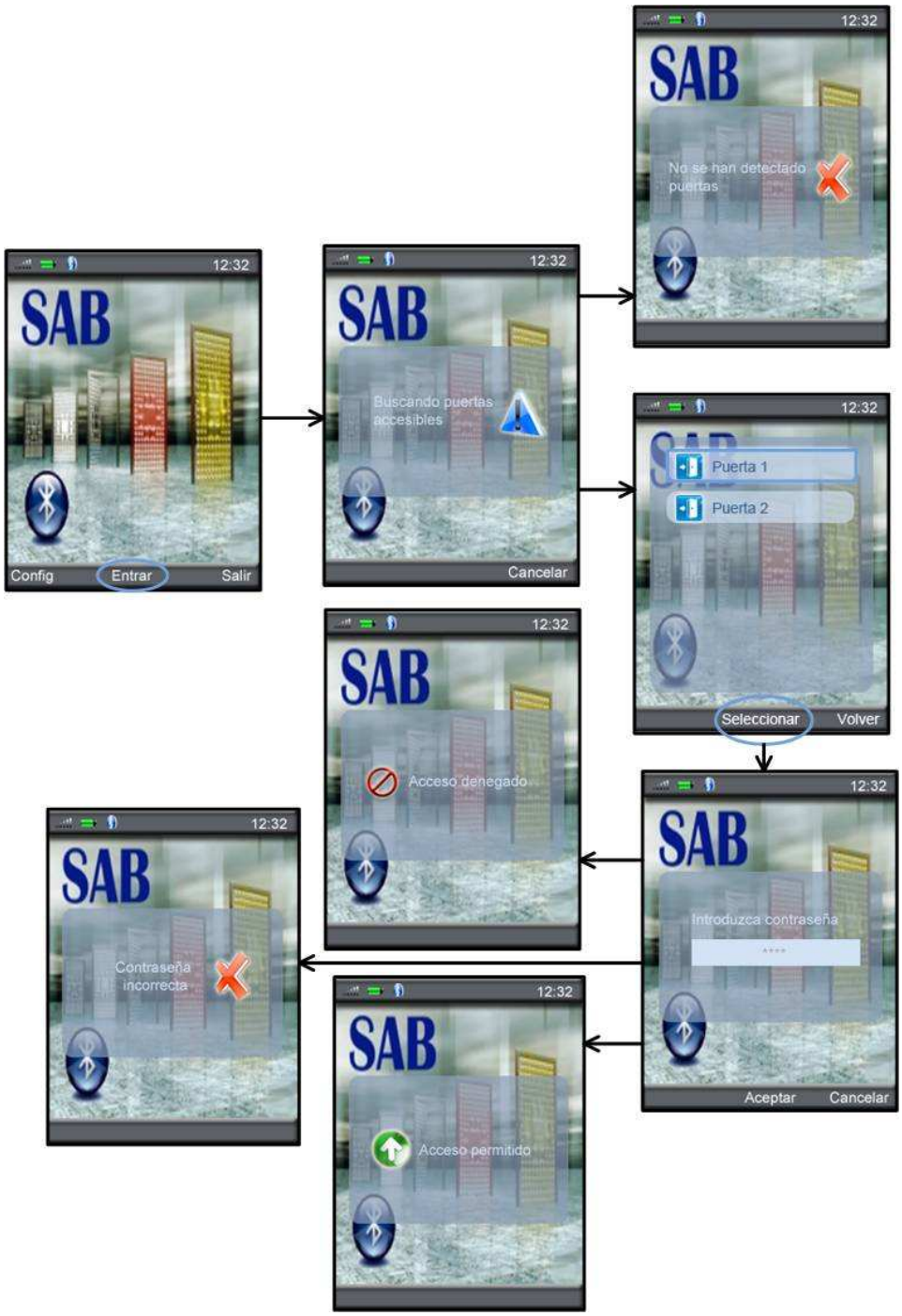


Figura 4.13. *Interfaz de la aplicación móvil*

Guardado de contraseña

Secuencia de las pantallas de la aplicación móvil para guardar la contraseña.

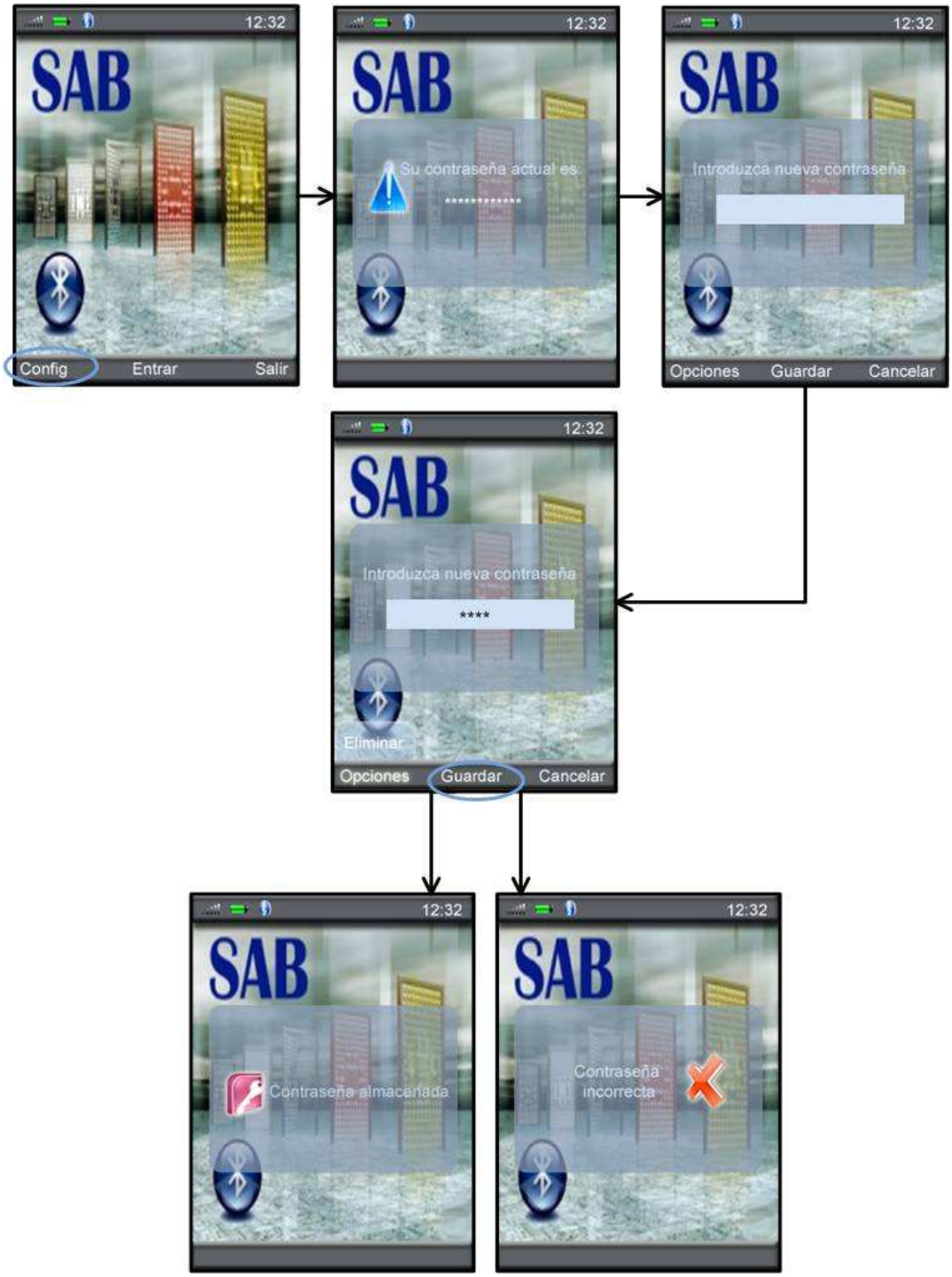


Figura 4.14. *Interfaz de la aplicación móvil*

Se pasó también por diferentes fases y mejoras hasta que se obtuvo la versión final. Las ampliaciones a destacar en ésta son:

- Pantalla LCD con conexión a las placas Arduino, para informar al usuario del acceso permitido/denegado.
- Incorporación al sistema de una cerradura electrónica y un relé, con las consiguientes conexiones necesarias.
- Ensamblaje de los componentes HW en una caja de PVC para tal efecto, y montaje de todo el sistema en una estructura de una ventana real.

Se puede ver en el ejemplo de ejecución siguiente que:

1. El móvil detecta el Arduino (puerta 1).
2. El Arduino está esperando recibir la información del móvil (MAC y contraseña).
3. El ServidorSk está esperando que el Arduino le envíe la información (MAC, contraseña y puerta)

Figura 4.15. *Prototipo final funcionando*

4. El ServidorSk (aplicación intermedia) accede al directorio LDAP y verifica que la MAC del móvil (00:1F:E4:2E:26:3E) tiene asociada la contraseña enviada y tiene acceso a la puerta solicitada (puerta 1).

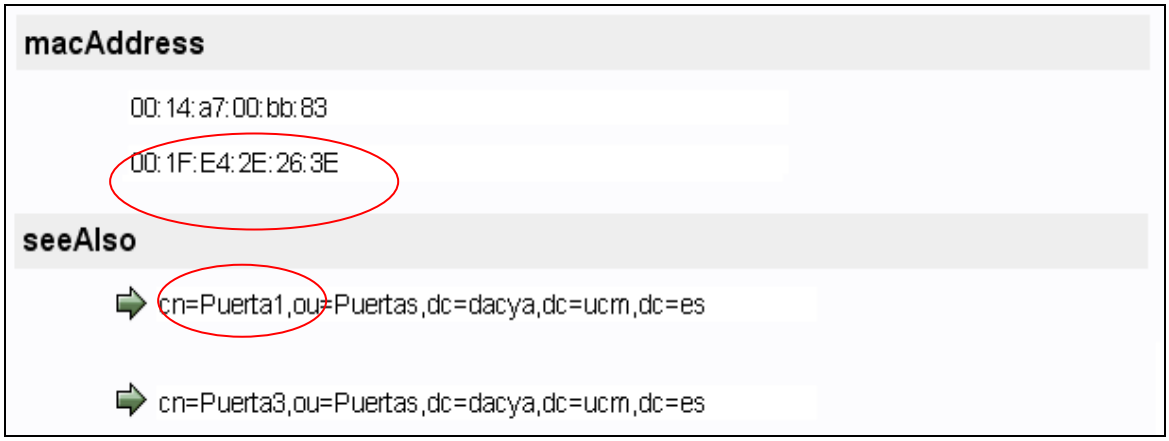


Figura 4.16. *Contenido directorio LDAP*

5. Una vez verificada la autorización del acceso, se envía el mensaje de "acceso permitido" al Arduino y de aquí al dispositivo móvil.

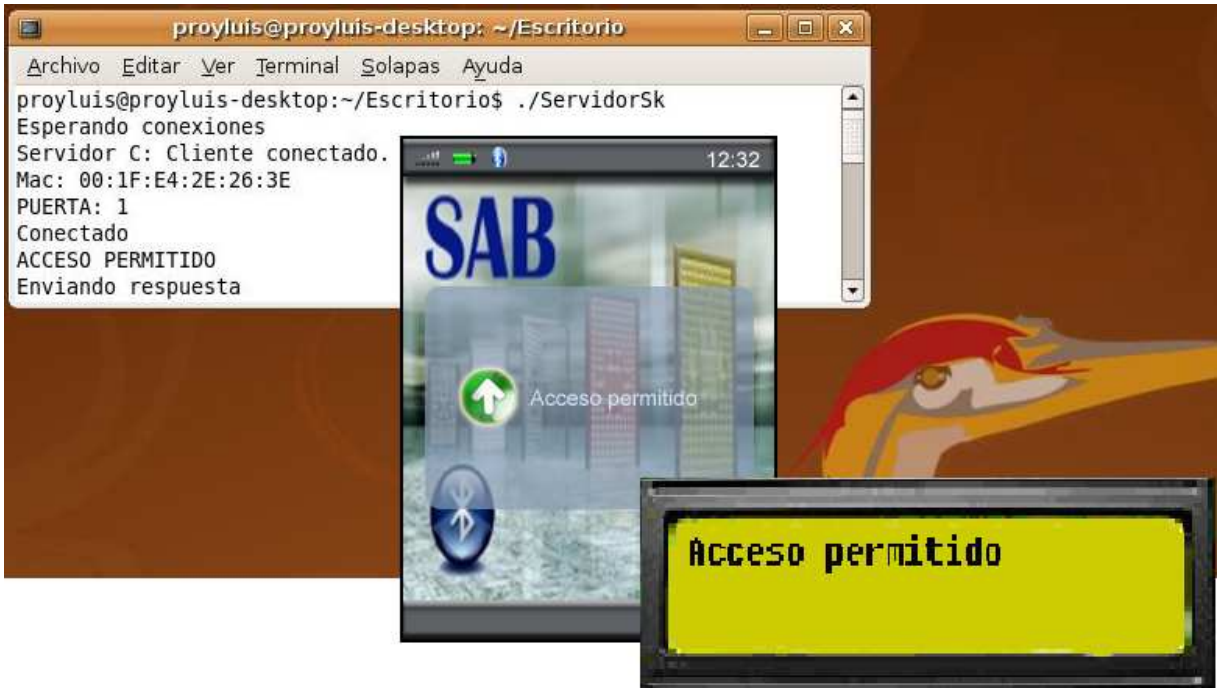


Figura 4.17. *Contenido directorio LDAP*

Figura 4.18. *Cerradura electrónica que es desbloqueada por el sistema*

Figura 4.19. *Caja en la que van ensamblados los componentes HW*

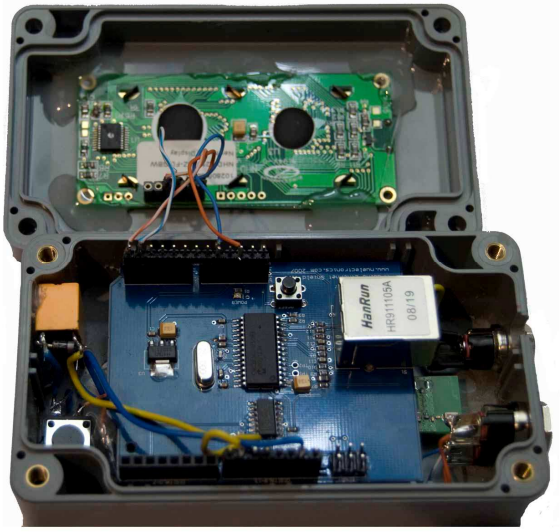


Figura 4.20. *Interior de la caja*

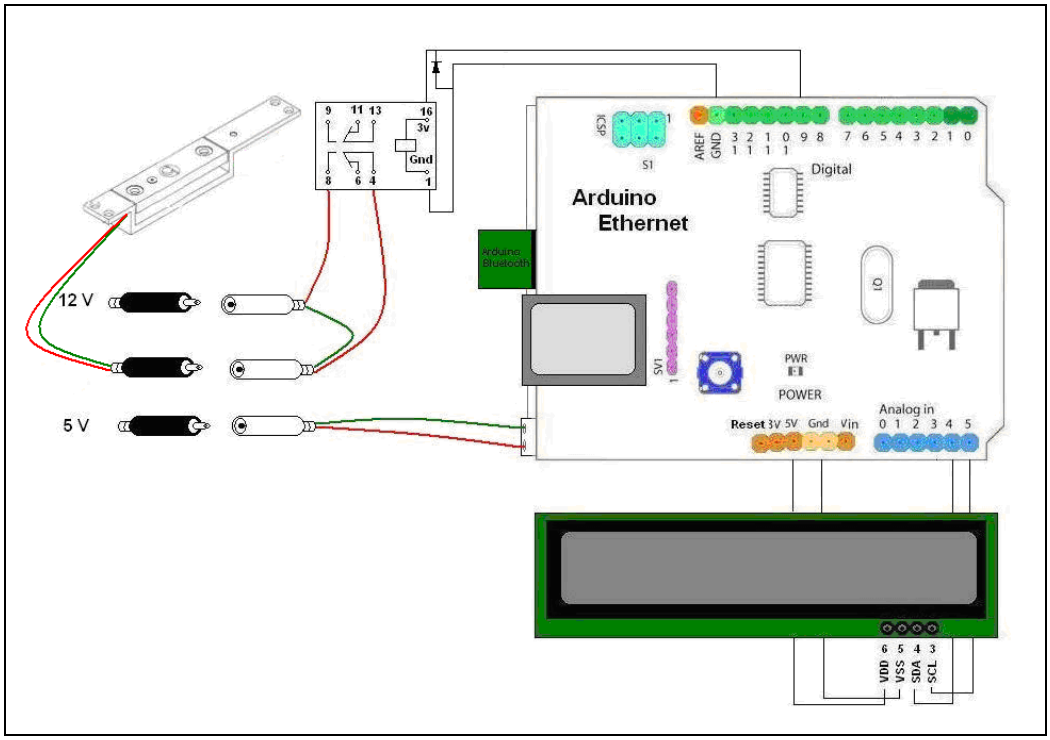


Figura 4.21. *Esquema de conexiones de los componentes HW*

Figura 4.22. *Ventana con la caja adosada*

Capítulo 5. Modelo de negocio

Este proyecto se ha desarrollado con una visión de producto y como tal debe llevar asociado un modelo de negocio que analice su posible comercialización. Concretamente el negocio es la comercialización del sistema de seguridad SAB y contempla diferentes posibilidades que vamos a estudiar en este capítulo.

5.1. Producto, servicios y clientes

PRODUCTO

- **Partes del producto final obtenido:**
  - Aplicación para el dispositivo móvil que hace de llave de acceso.
  - Sistema hardware que se asocia a cada puerta y cuyos principales componentes son las placas Arduino.
  - Aplicación intermedia que gestiona los accesos.
  - Sistema que almacena la información sobre los usuarios y sus permisos.
- **Opciones de compra del producto:**
  - Compra del producto completo, tal y como se ha especificado en esta memoria.
  - Que los Arduinos corran a cargo del cliente.
  - Que el sistema de almacenamiento utilizado sea uno del que ya dispone el cliente.
  - Sistema de almacenamiento creado por nosotros a petición del cliente.

SERVICIOS

Además del producto ofrecemos una serie de servicios que serán diferentes en función de la elección de compra y de los requerimientos del cliente.

- **Servicios:**
    - Instalación.
    - Administración.
    - Adaptación de la aplicación intermedia (si la opción de compra del producto implica la utilización de un sistema de almacenamiento del que ya dispone el cliente).
- **Restricciones de compra de servicios:**
    - El servicio de administración implica la compra de la instalación.

CLIENTES

Los clientes a los que va dirigido el producto y los servicios ofrecidos pueden dividirse en dos grupos principales:

- **Consumidor final:**
    - Entidades de la administración pública.
    - Empresas privadas.
    - Universidades.
    - Colegios.
    - Particulares.
- **Distribuidor intermediario:**
    - Empresas de seguridad.
    - Empresas de domótica.
    - Fabricantes de puertas.
    - Fabricantes de cerraduras.

5.2. Estudio de costes

Como hemos explicado antes ofrecemos varias opciones de compra del producto:

- **Paquete SAB:** conjunto de aplicaciones de software libre: aplicación Arduino, aplicación móvil y aplicación intermedia. [0 €]
- **Paquete 1**: Paquete SAB + Arduinos. Implícita la adaptación de la aplicación intermedia.
- **Paquete 2**: Paquete SAB + Arduinos. + LDAP estándar.
- **Paquete 3**: Paquete SAB + Arduinos. + LDAP/BBDD diseñada específicamente para el usuario.

Además está disponible la contratación de la instalación y la administración del sistema.

El precio final de cada paquete dependerá del número de Arduinos y de si se ha contratado instalación y/o administración. Vamos a especificar a continuación algunos precios desglosados:

Producto	Precio
Arduinos	90 € / unidad
LDAP estándar	100 €
LDAP/BBDD específica	A partir de 150 € (según especificación)

Tabla 5.1. *Precios*

5.3. Competitividad

En el mercado podemos encontrar múltiples sistemas de control de acceso físico, hablamos de los equipos de identificación personal basados en tarjetas de proximidad HID, tarjetas de código de barras, identificación de huella digital (biometría), tarjetas de banda magnética, teclados para ingresar números de usuario y contraseña, etc.

Vamos a realizar una comparativa de estos sistemas en relación al nuestro.

EQUIPOS DE IDENTIFICACIÓN	VENTAJAS	INCONVENIENTES
Basados en identificación de **huella digital** (Biometría)	▪ No requiere elementos de identificación adicionales. ▪ Confiere mayor seguridad	▪ Precio mucho mayor
Basados en **tarjetas de proximidad HID**	▪ Accesos más cómodos (no se requiere manipulación)	▪ Precio mayor ▪ Requiere la compra de elementos de identificación adicionales (las tarjetas) ▪ Confiere menor seguridad
Basados en **tarjetas de código de barras** o **tarjetas de banda magnética**		▪ Precio mayor ▪ Requiere la compra de elementos de identificación adicionales (las tarjetas) ▪ Confiere menor seguridad
Teclados	▪ No requiere elementos de identificación adicionales.	▪ Confiere menor seguridad

Tabla 5.2. *Comparativa de los sistemas de control de acceso físico*

Una vez expuestas las características de los sistemas de control de acceso de la comparativa, podemos valorar el sistema **SAB** por los motivos que se presentan a continuación:

▪ No requiere de la compra de elementos adicionales de identificación, pues para ello se utiliza un objeto cotidiano como es el teléfono móvil. ▪ Confiere un nivel de seguridad considerablemente alto, pues además de ser necesaria una contraseña, ésta se envía cifrada mediante un consistente algoritmo de encriptación. Por esto es más seguro que los sistemas de tarjetas, ya que éstas se pueden perder, o los teclados, que pueden dar lugar a que cualquiera que haya conseguido el código de acceso entre en el área restringida. ▪ Por lo comentado en el primer punto y por ser un sistema sencillo que utiliza una plataforma hardware de fuente abierta y bajo coste, el precio es menor que el de cualquiera de los sistemas de la comparativa. ▪ Como desventaja decir que es necesaria la manipulación del móvil para los accesos, pero cabe destacar que se contempla como futura ampliación la posibilidad de que funcione por proximidad.

Tabla 5.3. *Características del sistema SAB*

La comparativa se ha realizado únicamente a nivel del sistema físico, es decir, de sus componentes, su precio, su comodidad de uso y su nivel de seguridad porque en términos generales es lo único que tiene sentido comparar con respecto a los otros tipos de sistemas. Una vez hecho esto, podemos concluir que el sistema SAB no compite con los basados en biometría o similares por la complejidad, la seguridad y el alto precio de éstos. Sin embargo SAB sí puede competir y ser una buena alternativa a los basados en tarjetas y teclados.

A la hora de elegir la mejor opción entre todo lo que se ofrece en el mercado no basta sólo con valorar lo que se ha expuesto en la comparativa, sino que también hay que tener en cuenta las diferentes funciones que se ofrecen y que dependen del software y otros requerimientos, pero para esto hay que entrar en comparativas detalladas con productos concretos ofertados por una empresa específica y por ello hemos buscado productos similares al nuestro, productos que puedan parece competidores directos del SAB. Éste es el caso de la empresa Coltec, que ha desarrollado un sistema denominado BlueKey y que al igual que SAB se basa en la tecnología Bluetooth para abrir puertas mediante el teléfono móvil.

SAB		BlueKey	
=	Receptor abre puertas: 90 €	=	Receptor abre puertas: 102 €
=	Identificador personal : móvil	=	Identificador personal : móvil
=	Encriptación algorítmica segura	=	Encriptación algorítmica segura
=	Con un mismo dispositivo se abren todas las puertas que se quieran	=	Con un mismo dispositivo se abren todas las puertas que se quieran
+	Se pueden incorporar todos los usuarios que se quiera al sistema sin ningún coste adicional.	-	Tarjetas de códigos necesarios para incorporar usuarios con acceso a una puerta. Precio de una tarjeta de 5 códigos: 36,98€
-	Requiere Internet y una sistema de almacenamiento	+	No requiere Internet ni ningún accesorio más.

Tabla 5.4. *SAB vs BlueKey*

Una vez comparadas las características de ambos sistemas, podemos concluir que BlueKey está más orientada a viviendas de particulares, donde lo único que se busca es la apertura de un número reducido de puertas, sin conexión a Internet ni más elementos que el receptor de la puerta y el software que se instala en el dispositivo móvil. El sistema SAB, sin embargo, está orientado a instalaciones de mayor envergadura, como pueden ser las de una empresa o una universidad que quiere incorporar al sistema un gran número de puertas y usuarios y que ya posee sistemas informatizados con Internet e incluso con una base de datos que se pueda utilizar para los requerimientos del SAB. Por todo ello, en este caso, nuestro sistema sería la mejor opción, pues además de cubrir las necesidades del cliente, lo hace por un precio menor.

CONCLUSIÓN FINAL

Realizado este estudio creemos que nuestro producto podría tener un hueco en el mercado, pues aprovecha los elementos de los que ya disponen los clientes a los que va dirigido para incorporar a sus instalaciones un sistema de seguridad sencillo que ofrece las funcionalidades básicas que el cliente necesita con un bajo coste.

Conclusiones

En el presente documento se ha incluido información sobre las tecnologías aplicadas con la intención de crear un contexto adecuado en el que poder entender mejor el trabajo realizado durante el desarrollo del proyecto. Sin embargo no se puede plasmar todo el trabajo que implica diseñar y desarrollar un sistema como SAB, que incluye diferentes aplicaciones, cada una de ellas implementada con una tecnología diferente y en una plataforma distinta. Se ha tenido, por lo tanto, que realizar una labor de documentación considerable, y afrontar diversos problemas que han ido surgiendo durante todo el proceso. El trabajo con hardware fue algo nuevo para nosotros a lo que no estábamos acostumbrados y esto nos puso, en diferentes ocasiones, en la tesitura de afrontar dificultades con las que nunca habíamos lidiado. Hay por lo tanto gran cantidad de trabajo subyacente, en muchos casos de ensayo y error, para que un sistema de estas características lleve a cabo las funciones que finalmente se han conseguido.

Desde el primer momento le dimos a este proyecto un enfoque de producto, de producto real y comercializable, esto fue algo que siempre se tuvo presente y que, por lo tanto, influyó en la toma de decisiones, provocando algún que otro debate en más de una ocasión. Por todo ello, y una vez culminado el desarrollo, podemos decir que ahora somos mucho más conscientes del trabajo que conlleva realizar un proyecto de estas características y con este enfoque. Hemos cumplido los objetivos marcados y nos sentimos satisfechos del trabajo realizado.

Referencias

Java & Bluetooth

[1] Historia del Bluetooth
http://gbtcr.chileforge.cl/info_web/node75.html
http://es.wikipedia.org/wiki/Bluetooth

[2] Especificación y Arquitectura Bluetooth
http://spanish.bluetooth.com/Bluetooth/Technology/Works/Core_System_Architecture.htm
http://bluehack.elhacker.net/proyectos/bluesec/bluesec.html
http://www.tech-faq.com/lang/es/bluetooth.shtml

[3] J2ME
http://java.sun.com/javame/reference/apis/jsr118/
http://www.todosymbian.com/secart43.html
http://grasia.fdi.ucm.es/j2me/_J2METech/MIDP.html
http://leo.ugr.es/J2ME/INTRO/intro_8b.htm

[4] BlueCove
http://www.bluecove.org/

[5] API de BlueCove
http://www.bluecove.org/bluecove/apidocs/

[6] BlueCove. Código fuente de la aplicación de demostración
http://demo.jorgeivanmeza.com/Java/BlueCove/RemoteDeviceDiscovery/

Aplicación intermedia

[7] Sockets
http://www.icesecurity.org/programacion/Sockets/sockets.html

[8] ldapc++
http://www.openldap.org/devel/cvsweb.cgi/contrib/ldapc%2b%2b/?hideattic=1&sortbydate=0

[9] Desencriptación
http://jaliyacgl.blogspot.com/2007/07/august-8th-report.html

Arduino

[10] Ethernet
http://www.nuelectronics.com/estore/index.php?main_page=product_info&cPath=1&products_id=4

[11] Bluetooth
http://arduino.cc/en/Main/ArduinoBoardBluetooth

[12] LCD
http://store.fungizmos.com/index.php?main_page=product_info&cPath=70&products_id=210

Empresas

[13] Producto con el que hemos realizado una comparativa
http://www.bluekey.es/index.php

[14] Empresas de seguridad que nos han servido para documentarnos y estudiar el mercado
http://www.blauden.com/sobre-nosotros
http://www.brivas.com/seguridad/calf.htm
http://www.asistaweb.cl/
http://www.alarmas-seguridad.es/casmar-electronica-distribuidor-sistemas-seguridad/2-31-3-31.htm

Móviles compatibles

[15] Información sobre los móviles que disponen de la librería JSR177
http://mobilezoo.biz/search.php?action=search&search=jsr177

Anexo
Móviles compatibles

En la siguiente tabla se especifican los modelos de móviles que actualmente soportan la API **JSR-177** y, por lo tanto, son compatibles con el sistema desarrollado en este proyecto.

#	Modelo	#	Modelo
1	Nokia 3250	57	Samsung SGHE210
2	Nokia 6280	58	Samsung SGHU600-Orange
3	Nokia E61	59	Samsung SPHM620
4	Nokia E70	60	Samsung SGHE490
5	Nokia N71	61	Samsung SGHJ600
6	Nokia 6630	62	Samsung SGHU300
7	Nokia N73	63	Samsung SPHM510
8	Nokia E60	64	Samsung SGHU100
9	Nokia N93	65	Samsung SGHE950
10	Nokia E50	66	Samsung SGH-L770
11	Nokia N91	67	Samsung SGHJ600E
12	Nokia E62	68	Samsung SPHM520
13	Nokia 5500d	69	Samsung SGH-i450
14	Nokia N80	70	Samsung SGH-i550
15	Nokia N95	71	Samsung SGH-i520
16	Nokia N75	72	SonyEricsson W850i
17	Nokia N76	73	SonyEricsson W950i
18	Nokia E90	74	SonyEricsson W910i
19	Nokia E65	75	SonyEricsson Z750i
20	Nokia E61i	76	SonyEricsson K850i
21	Nokia N93i	77	SonyEricsson K630c
22	Nokia 6110 Navigator	78	SonyEricsson V640i
23	Nokia N92	79	SonyEricsson K630i
24	Nokia 6290	80	SonyEricsson K660i
25	Nokia 6110Navigator	81	SonyEricsson W908c
26	Nokia 6555	82	SonyEricsson V660i
27	Nokia E90	83	SonyEricsson W890i
28	Nokia 5700	84	SonyEricsson K850c
29	Nokia 6267	85	SonyEricsson K858c
30	Nokia 6120c	86	SonyEricsson K123i

31	Nokia N77	87	SonyEricsson W760i
32	Nokia N95_8GB	88	SonyEricsson A120i
33	Nokia 6500s	89	Siemens SXG75
34	Nokia 7500	90	Siemens MTK
35	Nokia N81	91	Siemens ME4SE
36	Nokia 5310XpressMusic	92	Siemens MicroEmulator
37	Nokia N82	93	Siemens MicroEmulator-2.0
38	Nokia 6500c	94	Siemens E71
39	Nokia E51	95	Siemens EF81
40	Nokia 5610d	96	Siemens EL71
41	Nokia 7900	97	Benq EF81
42	Nokia 6555	98	Benq SL91
43	Nokia 6121c	99	Benq M7
44	Nokia 8800e	100	LG LG550
45	Nokia generic	101	LG LG800
46	Samsung SPHA920	102	LG CU720
47	Samsung SPHA900	103	LG LX550
48	Samsung SPHM500	104	Sanyo SCP7050CA Configuration
49	Samsung SPHA940	105	Sanyo SCP7050CA
50	Samsung SPHA900P	106	Vertu AscentTi
51	Samsung SGHU600	107	Vodafone V702NK (Nokia 6630)
52	Samsung SGHE840	108	Vodafone SamsungSGH-i520
53	Samsung SGHE570	109	Vodafone V804NK
54	Samsung SGHE840-Orange	110	Vodafone SamsungSGH-i550
55	Samsung SGHE790	111	Sun Wireless Toolkit
56	Samsung SGHE576	112	Unknown unknow